INNOVATIONS IN LAST-MILE DELIVERY SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Luis Damián Reyes Rodríguez

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
H. Milton Stewart School of Industrial and Systems Engineering

Georgia Institute of Technology

May 2018

INNOVATIONS IN LAST-MILE DELIVERY SYSTEMS

Approved by:

Dr. Martin Savelsbergh, Advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Alan L. Erera, Advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Alejandro Toriello
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Basak Kalkanci
Scheller College of Business
Georgia Institute of Technology

Dr. Andy Sun
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: December 13, 2017

Para Kelly, *por supuesto.*

ACKNOWLEDGEMENTS

This dissertation is dedicated to my partner, Kelly Ventura, as an acknowledgement of the sacrifice she made when she put her career on hold to join me in Atlanta. Her unending encouragement, love and care have kept me going throughout all these years. To her I shall be forever grateful.

I am also immensely thankful to my parents, Susi Rodriguez and Sigfrido Reyes, who raised me with love, planted the seeds of my values and convictions, and continue to have my back on all my endeavors. To my sister, Diana, whom I admire so much, and whose graduation I had to miss. To my family and friends all, for their affection.

This dissertation condenses a large body of discussions and intellectual exchanges between many people. Martin Savelsbergh, with whom I worked since my first week in the program, spent countless hours introducing me to new concepts and tools, as well as revising everything I wrote, always showing the right amount of patience and tact while challenging me to become more rigorous, emphatic, and clear in my research. Alan Erera, who was closely involved from the second year onwards, always managed to make the time in his heavy agenda to provide sharp insights, suggest important improvements, and offer general advice and encouragement: I could not have thought of a better pair of thesis advisors. Alejandro Toriello, who was deeply involved as a co-author in the paper that would become the first part of the thesis, also played an important role in determining the direction of my research interests (through our research collaboration, and through his lectures in Logistics Systems Engineering). Committee members Basak Kalkanci and Andy Sun oversaw the process with the utmost diligence. And the research team at Grubhub; fellow students Mathias Klapp, Luke Marshall, and Ramón Ahuad; as well as anonymous referees, all contributed to enrich this thesis. To all of them I owe them my highest respect and gratitude.

The professors, students and workers at ISyE constituted a great community and environment for my studies and research work. Special thanks go to Prof. Chen Zhou, for his wise advice during my first year in the program. And to the people at the ISyE technical helpdesk and academic office, for their excellent service.

I will never forget the solidarity and warmth of all the friends that Kelly and I have made in Atlanta. I highlight the immense help that friends Mathias Klapp, Francisca Otero, Asteroide Santana, Denise Batista, Tony Yaacoub, and Tom Gagné have provided for our survival these years in Atlanta.

Finally, I reflect on the influence exerted on my formation by all the teachers, mentors and fellow classmates through my years at Externado de San José, the University of El Salvador, and the University of Illinois, with many of whom I will not cross paths again. I guess I can only pay it forward.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In the last decade, tied to the rise of e-commerce activities, direct-to-consumer delivery operations have grown at a rapid pace in metropolitan areas across the world, and expectations on service standards become higher and higher every year. The economic, social and environmental sustainability of last-mile logistical operations poses a formidable societal challenge, demanding major organizational changes and technological inventions in transportation, the success of which will critically depend on the availability of appropriate optimization tools.

In this thesis, we study two recent innovations in last-mile delivery from an optimization perspective: i) *roaming delivery* systems, where the customer orders are delivered to the trunk of their cars, as opposed to traditional home delivery; and ii) dynamic delivery systems, where vehicles deliver goods locally from an origin depot (or, perhaps a small number of origin depots) to customer locations, and the requests for delivery arise during the vehicle operating period, which, if accepted, must be satisfied within a service window (*e.g.*, meal delivery) or by the end of the day (*e.g.*, same-day delivery).

We introduce the *vehicle routing problem with roaming delivery locations* and the *meal delivery routing problem*, to formalize and conduct a systematic study of the essential features of these systems, and then develop heuristics and optimization-based tools that enable their successful deployment. On a more basic level, we contribute to the understanding of the relation between cost and service quality objectives in dynamic delivery systems, through the study of a series of models with a highly simplified geometry, exploring the structure of optimal solutions and providing efficient algorithms to solve some vehicle routing and demand management (order acceptance strategies used when the system is overwhelmed by demand) problems.

In Chapter 2, after defining the *vehicle routing problem with roaming delivery locations*

(VRPRDL), we develop construction and improvement heuristics based on problem-specific techniques. Results from a computational study suggest that *roaming delivery* systems can have a positive economic and environmental impact, as their deployment can lead to a significant reduction in total distance traveled, especially when used in conjunction to traditional home delivery.

In Chapter 3, we begin our investigation of dynamic delivery systems with a study on the complexity of single depot dispatching problems in which a delivery to a customer must occur within a pre-specified time after the customer places the order. Thus, each order has a release date (the earliest time that the order can be dispatched on a route) and a service guarantee that implies a deadline (when the order needs to be delivered). We show that single and multiple vehicle variants where customers are located on a half-line can be solved to optimality in polynomial time.

In Chapter 4, we define the *meal delivery routing problem* and propose optimization--based algorithms tailored to solve the courier assignment (dynamic vehicle routing) and capacity management (offline shift scheduling) problems in meal delivery systems. Computational experiments on instances with realistic size, geography, urgency and dynamism (based on data provided by our industry partner, Grubhub) demonstrate that our algorithmic ideas can be valuable in real-world implementations.

In Chapter 5, we conduct an initial exploration of demand management interventions that can be used when dynamic delivery systems are overwhelmed by demand. We specifically assess the effectiveness of temporarily reducing the coverage area of the system (rejecting orders farther away than a given radius from the depot). We extend the models from Chapter 3 to add order acceptance decisions and prove that some variants can still be solved in polynomial time. Then we propose a series of dynamic algorithms and test them through simulation. Results illustrate that successful heuristics for "same-day" delivery may not be well suited for "next hour" delivery.

CHAPTER 1
INTRODUCTION

1.1 Last-mile delivery innovations

During the last decade we have witnessed the revolution of internet-based technologies, among them e-commerce, which has grown swiftly from about 3.5% of total US retail sales in the second quarter of 2008 to 8.9% in the second quarter of 2017, according to the US Census [1], and whose growth seems to be accelerating. Estimations that exclude groceries from the totals show that e-commerce is already becoming dominant: 2016 marked the first time that online purchases surpassed in-store purchases of retail items (excluding groceries) in the United States [2, 3]. This explosive growth has brought about a significant rise in direct-to-consumer deliveries, which now represent more than half of the last-mile logistical activities in some developed economies, like Germany [4].

Contributing to this rise are online retailers' offers of ever faster service, spearheaded by Amazon, that consumers have become accustomed to: increasingly common same-day, 2-hour and even 1-hour delivery promises are projected to constitute about 20% to 25% of the market by 2025 [4], disrupting the monopoly on 'instant gratification' that brick-and-mortar stores have historically enjoyed. In response, more and more traditional retailers, from small corner bakeries to Walmart, have begun to embrace the use of innovative delivery solutions [5] and are experimenting with fulfillment strategies like "buy-online, deliver-from-store" [6].

To keep up with expected demand, the landscape of last-mile delivery providers is transforming as well [7]. Major logistics companies like UPS and FedEx, specialized in next-day delivery, are gradually adapting to use their infrastructure, know-how and pres-

tige, and capture their share of the emerging same-day market. Simultaneously, significant investments have been made in start-ups bidding to satisfy the appetite for same-day and instant delivery of retail purchases (*e.g.*, Instacart and Deliv) and restaurant meals (*e.g.*, Grubhub and Delivery Hero), all the while leading companies in related industries are leveraging their position and deploying their own last-mile delivery services, like Amazon Flex and Uber Rush.

However, the sustainability of this frenetic growth is not guaranteed yet: if the higher volumes of e-commerce activities rely on inefficient last-mile delivery systems, the result is likely to be a substantial increase in vehicle miles traveled, especially in residential urban areas, which not only results in high costs, but also in increased emissions, increased congestion, and a consequent decrease in urban quality of life. In other words, if the new service models are not supported by the appropriate logistics processes and optimization technology, these may well end up intensifying rather than ameliorating already existing economic, social and environmental problems.

This dissertation is a contribution in the development of novel algorithmic technology that can help support the sustainable implementation of two particular innovations in last-mile logistics: roaming delivery systems, and dynamic delivery systems.

1.1.1 Roaming delivery systems

The first part of the thesis is dedicated to studying the routing problem that emerges from the use of secure technologies to remotely control access to a vehicle's storage compartments, effectively transforming the trunk of a car into a roaming personal mailbox. By delivering to the trunk of a customer's car rather than to the customer's home, delivery may take place closer to the fulfillment center and is much more likely to be successful. The technology to facilitate this mode of delivery is currently offered by several auto manufacturers, and some of the partnerships testing the concept are Volvo and Urb-it [8],

Daimler, DHL and Amazon [9], and Audi, DHL and Amazon [10].

The possibility of trunk deliveries leads to a fundamentally different variant of the well-known *vehicle routing problem* (VRP). The VRP has been extensively studied since its introduction in the early 1950s [11], and the family of VRP variants is huge and steadily growing[12], encompassing the VRP with time windows, the VRP with pickups and deliveries, the dynamic VRP, the VRP with stochastic demands, and the split delivery VRP, to name just a few. But in all these problem variants, the customer's delivery location, *i.e.*, the location where the delivery occurs, is given (even if the decision maker may not have perfect information about it). When deliveries are made to a customer's car, this constancy disappears, because the car will likely be parked in different locations during the planning horizon, *e.g.*, at work, at the mall, at church, at the kids' soccer practice, and so on, and a delivery location (and thus delivery time) must be chosen by the service provider. To capture this new feature, we propose the *VRP with roaming delivery locations* (VRPRDL) as a canonical optimization model.

The VRPRDL is closely related to the VRP with time windows and the Generalized VRP. In the vehicle routing problem with time windows (VRPTW), each customer requiring a delivery has a time window during which the delivery can take place [13, 14, 15, 16]. (A vehicle is usually allowed to wait at a customer's location until the start of the time window.) In the generalized vehicle routing problem (GVRP), the set of customers is partitioned into clusters and each cluster has a given demand. The objective is to construct a minimum cost set of delivery routes serving one of the customers in each cluster such that total demand of the customers served by a single vehicle does not exceed the vehicle capacity [17]. We are aware of only a single paper [18] that considers the generalized vehicle routing problem with time windows (GVRPTW), *i.e.*, the variant that combines the characteristics of the VRPTW and the GVRP. By splitting a single customer in the VRPRDL into multiple customers, one for each of the locations visited in the customer's

geographic profile, we obtain a special case of the GVRPTW. It is a special case, because the time windows of the customers in a cluster do not overlap.

1.1.2 Dynamic delivery systems

The second part of the thesis is devoted to studying dynamic *delivery* systems, paying particular attention to meal delivery applications.

In dynamic delivery systems, vehicles deliver goods locally from an origin depot (or, perhaps a small number of origin depots) to customer locations, and the requests for delivery arise during the operating period; *e.g.*, [19]. The defining characteristic in the routing problems that arise in this context is that when customer requests become known, they not only have a deadline, which specifies the latest time the delivery can be made, but also a *release date*, which specifies the earliest time the goods to be delivered are ready to be dispatched from the depot. The ready time can be the time that a request is made, but it can also be later, for example, to account for order picking and staging (if the goods are stocked at a warehouse), or order processing and preparation (if the goods are made-to-order, as in the case of restaurant meals).

An important class of dynamic delivery systems is found in the restaurant meal delivery industry. On-demand meal-ordering platforms – online marketplaces where diners order their favorite cravings from an array of restaurants – are growing at a fast pace, and the volume of meal delivery operations is rising quickly [20], opening up new economies of scope, scale, and density, which emerging providers are aiming to capitalize through the deployment of meal delivery networks. Such systems face complex capacity planning problems and increasingly large dynamic pickup and delivery problems that must be solved in (near) real-time [21, 22, 23, 24]. Due to the high dynamism and extreme urgency of arriving orders [25], meal delivery represents the ultimate challenge in last-mile logistics: a typical order is expected to be delivered within an hour (much less if possible),

and within minutes of the food becoming ready, thus reducing consolidation opportunities and imposing the need for more vehicles operating simultaneously and executing shorter routes (which can be costly).

Successful dynamic delivery networks must be able to respond to wide, and often abrupt, swings in demand both in spatial and time dimensions. In an attempt to achieve the desired responsiveness without the costs linked to employing a sufficiently large permanent fleet of vehicles (and full-time couriers), companies are resorting to "digital marketplace" business models – where the supply of couriers who are independent contractors [26] is controlled by economic incentives (*e.g.*, per-unit payments, bonuses, service ratings). This strategy, first explored in the context of taxi and ride-hailing services, externalizes fixed costs (to couriers) and enhances the ability of the system to plan and control capacity levels over time and geography in sync with demand fluctuations.

However, it is worth emphasizing how full reliance on a fleet of independent contractors establishes a fundamentally different operating environment than that of traditional vehicle routing applications. In exchange for accepting some of the risks and costs associated with demand uncertainty, couriers are entitled to a significant degree of autonomy [27], thereby adding yet another layer of complexity in the design of appropriate optimization technology: company drivers *will* implement instructions from central planning, whereas independent contractors *might* do so. As companies give up their ability to tightly direct and enforce delivery couriers' schedules and routes, the temporal and geographic distribution of capacity cannot be anticipated with full certainty. This, combined with dynamic, uncertain, demand, makes the design of algorithms for fleet planning and control a formidable challenge.

Aside from improvements in scheduling of the courier supply, better service quality in dynamic delivery systems may be induced by influencing demand through an array of possible incentives and interventions, which may involve: delivery fees, minimum pur-

chase policies, delivery time estimates, web-search suggestions, changes in the service coverage area, or even localized service shut-downs. While demand management tactics being explored in the context of ride-sharing may seem directly applicable to dynamic delivery, some cases like meal delivery illustrate an important new feature, namely *demand substitution* opportunities: customers shopping for their meals are often flexible in their choice of restaurant. Understanding different demand management tactics in dynamic delivery systems from the perspective of logistical performance represents a novel and rich research opportunity.

1.2 Contributions of the thesis

In Chapter 2, after formally defining the Vehicle Routing Problem with Roaming Delivery Locations, we develop an effective heuristic for its solution. Though partly based on known techniques, the heuristic includes innovations, namely the computationally efficient management of a set of feasible solutions during construction and local search procedures, and the use of an efficient dynamic programming algorithm to optimize parts of the heuristic solution at appropriate times. Afterwards, we conduct an extensive computational study to assess and quantify the benefits of roaming delivery, either as a replacement of home delivery or in conjunction with home delivery. The study reveals that, depending on the geography and the assumptions about daily travel itineraries of customers, the benefits can be significant, in certain settings resulting in a reduction of the total distance traveled of more than 50% (and the concomitant reduction in emissions, congestion, etc.). The main contents of this chapter have been recently published previously in [28].

In Chapter 3, we investigate the computational complexity of dynamic delivery problems in some simple settings. This theoretical detour is motivated by the fact that, although dynamic vehicle routing problems have received a significant amount of atten-

tion in the literature (see, for example, the excellent surveys by Berbeglia, Cordeau, and Laporte [21] and Pillac, Gendreau, Guéret, and Medaglia [29], and [22]), the class of dynamic delivery problems is just beginning to be explored. While it is clear that, in their full generality, routing problems with release dates and deadlines subsume well-known NP-hard problems, we show that when all delivery locations are on a half-line or, more broadly, on a "star" network, some interesting perfect-information variants can be solved in polynomial time. These results are a step forward in the research direction initiated by the work of Archetti, Feillet, and Speranza [30], with the sight ultimately set on obtaining lower bounds to evaluate the performance of dynamic delivery heuristic algorithms. The results presented in this chapter have been recently published in [31].

In Chapter 4, we introduce the Meal Delivery Routing Problem to formalize and study this important class of dynamic delivery operations. We develop optimization-based algorithms tailored to solve the courier assignment (dynamic vehicle routing) and capacity management (offline shift scheduling) problems in meal delivery systems. We conduct extensive computer simulations to investigate the effectiveness of our approach, under different realistic system conditions and instance characteristics. Results demonstrate that our algorithmic ideas can be valuable in real-world implementations. To the best of our knowledge, our instances are the first set derived from historical meal delivery data (provided by our industry partner, Grubhub) to be released in the public domain.

In Chapter 5, we conduct an exploration of the complexity and relative performance of a specific demand management intervention for meal delivery systems, where the goal is to manipulate the service coverage radius of restaurants in the system to efficiently preserve service quality in the face of overwhelming demand over an operating period. Concretely, we develop simplified models on a half-line geometry and solve some perfect information variants in polynomial time using dynamic programming, to then evaluate different heuristic procedures in a dynamic simulation environment.

Part I

Roaming Delivery Systems

CHAPTER 2

THE VEHICLE ROUTING PROBLEM WITH ROAMING DELIVERY LOCATIONS

2.1 Introduction

As mentioned in chapter 1, the explosive growth of e-commerce poses a major challenge in the so-called "last mile" of the supply chain, the final delivery of goods to the consumer. Higher volumes of direct-to-consumer e-commerce, tied to inefficient last-mile delivery systems, may end up leading not only to high costs but also to exacerbated social and environmental problems like pollution and congestion. The situation is worsened when customers must sign upon delivery (as is the case in most European countries) and multiple visits are made due to missed deliveries.

The negative impacts of an increase in vehicle miles traveled can be mitigated by deploying environmentally-friendly vehicles, and many companies are investigating (or are forced by impending regulations to investigate) such options. However, improvements in operational practices can also make a large contribution – for example, Walmart attributes 39% of their efficiency gains during 2005-2015 to improvements in routing [32] – and should therefore be explored as well; that is the focus of this chapter. The efficiency of last-mile delivery can be improved in at least two ways: by decreasing the distance from the fulfillment center to the delivery locations (which will increase delivery location density) and by reducing the number of missed deliveries. Both of these can be achieved through the use of parcel box deliveries – an increasingly popular option, e.g. United Parcel Service of America, Inc. [33] – and the use of trunk deliveries, which are the focus of this chapter.

By delivering to the trunk of a customer's car rather than to the customer's home, de-

livery may take place (if carefully timed) at a location closer to the fulfillment center, or closer to other delivery locations. Furthermore, there are two main reasons why companies choose not to allow delivery at the door when the customer is not at home - inclement weather and theft risks - and the adoption of trunk delivery greatly reduces both of them, thus eliminating the need for the dreaded "we missed you" notes.

The innovative idea of trunk delivery can be traced back to start-up company Cardrops (`www.cardrops.com`), which in 2012 first attempted to tackle the technical challenges through the use of a device with GPS-tracking abilities and control of the trunk lock [34] installed inside customers vehicles. Shortly afterwards, the security and communication technologies enabling this mode of delivery began to be seamlessly integrated as features of the latest car models – a move part of a broader push by car manufacturers to make physical keys a thing of the past [35]. A proof of concept was introduced by Volvo at the 2014 Mobile World Congress [36], and pilot studies followed suit [37]:

> Holiday shopping will be pleasant for those in Gothenburg, Sweden. Volvo kickstarted its own commercially-available In-car Delivery service that aims to liberate consumers who choose to shop online.
>
> With the new service, online shoppers can have the package delivered directly to their cars, even if they are away from it. The whole delivery process becomes feasible with a unique one-time-use digital key that couriers get to unlock the client's vehicle.

Since then, important car-manufacturing and logistics companies have started partnerships to experiment with the concept, among them Volvo and Urb-it [8], Daimler, DHL and Amazon [9], and Audi, DHL and Amazon [10]:

> "Our final attempt failed, your package can be picked up at our service center." With Audi connect easy delivery, a future service from Audi connect,

> this message will be a thing of the past – making shopping online even more convenient. If the Audi owner agrees to the tracking of their automobile for the specific delivery time frame, the DHL driver handling the parcel receives a digital access code for the trunk of the customer's vehicle. It can be used one time only for a specific period of time and expires as soon as the luggage compartment has been closed again. Similarly, Audi connect easy delivery customers will also be able to send letters and parcels from their own car in the future.

From a methodological perspective, the possibility of trunk deliveries leads to a fundamentally different variant of the well-known *vehicle routing problem* (VRP). The VRP has been extensively studied since its introduction in the early 1950s, and there is a plethora of VRP variants, including the VRP with time windows, the VRP with pickups and deliveries, the dynamic VRP, the VRP with stochastic demands, and the split delivery VRP, to name just a few. In all these problem variants, the customer's delivery location, i.e. the location where the delivery occurs, is given (even if the decision maker may not have perfect information about it). When deliveries are made to the trunk of a customer's car, this constancy disappears, because the customer's car will likely be in different locations during the planning horizon, e.g. at work, at the mall, at church, at the kids' soccer practice, and so on, and a delivery location (and thus delivery time) must be chosen by the service provider. We propose the *VRP with roaming delivery locations* (VRPRDL) as a canonical optimization model to capture this new feature.

The contributions of the research reported in this chapter are twofold. First, we introduce an interesting and practically relevant new variant of the VRP, and develop an effective heuristic for its solution. Though partly based on known techniques, the heuristic includes innovations, namely the computationally efficient management of a set of feasible solutions during construction and local search procedures, and the use of an efficient

dynamic programming algorithm to optimize parts of the heuristic solution at appropriate times. Second, we conduct an extensive computational study to assess and quantify the benefits of trunk delivery, either as a replacement of home delivery or in conjunction with home delivery. The study reveals that, depending on the geography and the assumptions about daily travel itineraries of customers, the benefits can be significant, in certain settings resulting in a reduction of the total distance traveled of more than 50% (and the concomitant reduction in emissions, congestion, etc.).

The remainder of the chapter is organized as follows. We close this section with a brief review of relevant literature. Section 2.2 then formally introduces the problem and gives an integer programming formulation for it. Section 2.3 discusses the optimization of a route for a fixed sequence of customers, an important sub-problem. Section 2.4 proposes various constructive and improvement heuristics, which we compare computationally in Section 2.5. This section also presents the results of an extensive computational study assessing and quantifying the benefits of trunk delivery. Section 2.6 closes with some final remarks, while an appendix contains more detailed technical material.

2.1.1 Relevant literature

There is a huge and ever-expanding body of literature on vehicle routing and scheduling problems. For conciseness, we provide only a few references for each of the vehicle routing and scheduling variants relevant to our research.

In the canonical vehicle routing problem (VRP), a set of geographically dispersed customers has to be visited to satisfy their demand. The objective is to construct a minimum-cost set of delivery routes serving all customers such that the total demand of the customers served by a single vehicle does not exceed the vehicle capacity [11, 12]. In the vehicle routing problem with time windows (VRPTW), each customer requiring a delivery has a time window during which the delivery can take place [13, 14, 15, 16]. (A vehicle

is usually allowed to wait at a customer's location until the start of the time window.) In the generalized vehicle routing problem (GVRP), the set of customers is partitioned into clusters and each cluster has a given demand. The objective is to construct a minimum cost set of delivery routes serving one of the customers in each cluster such that total demand of the customers served by a single vehicle does not exceed the vehicle capacity [17].

We are aware of only a single paper [18] that considers the generalized vehicle routing problem with time windows (GVRPTW), i.e., the variant that combines the characteristics of the VRPTW and the GVRP. By splitting a single customer in the VRPRDL into multiple customers, one for each of the locations visited in the customer's geographic profile, we obtain a special case of the GVRPTW. It is a special case, because the time windows of the customers in a cluster do not overlap.

There are other routing models that share some similarities with the VRPRDL. The most important one is the time-dependent vehicle routing problem (TDVRP) where the cost (or travel time) to go from one location to another depends on the departure time [38, 39, 40]. In the VRPRDL, the travel time from one customer to the next also depends on the time of departure, as the departure time defines the location where the departure takes place. However, contrary to the TDVRP, the travel time is not uniquely defined by the departure time, because the next customer may be reached in different locations.

2.2 The VRP with roaming delivery locations

The VRP with Roaming Delivery Locations (VRPRDL) can be formally defined as follows. Let $(N \cup \{0\}, A)$ denote a complete directed graph with node set $N \cup \{0\}$ and arc set A, where node 0 represents the depot and N represents a collection of locations of interest. Each arc $a \in A$ has an associated travel time t_a and cost w_a, both of which satisfy the triangle inequality. C represents the set of customers that require a delivery during the

planning period $[0, T]$. The delivery for a customer $c \in C$ is characterized by a demand quantity d_c and geographic profile $N_c \subseteq N$ that specifies where and when a delivery can be made. Specifically, each location $i \in N_c$ has a non-overlapping time window $[a_i^c, b_i^c]$ during which the customer's vehicle is at i. By duplicating locations, we may assume $N_c \cap N_{c'} = \emptyset$ for different customers c, c', and to simplify notation we also assume $|N_c| = k$ for all customers c. Each customer's time windows naturally imply an ordering of locations $i_1^c, \ldots, i_k^c \in N_c$; we assume that $i_1^c = i_k^c$ represent the customer's home location and that the time windows satisfy

$$a_1^c = 0; \qquad b_k^c = T; \tag{2.1a}$$

$$a_\ell^c = b_{\ell-1}^c + t_{i_{\ell-1}^c, i_\ell^c}, \quad \ell = 2, \ldots, k. \tag{2.1b}$$

In particular, condition (2.1b) indicates that when a customer's vehicle moves from one location to another, it incurs the same travel time as the delivery vehicles and is unavailable during this time; we further explain the reason for this assumption in Section 2.3 below. A set V of homogeneous vehicles with capacity Q is available to make deliveries; vehicles start and end their delivery routes at the depot. The goal is to find a set of delivery routes (a sequence of customer locations) and delivery times such that every customer receives a single delivery during the planning period, the total demand delivered on a delivery route does not exceed Q, the total travel and waiting time of a delivery route does not exceed T, and the total cost is minimized. The traditional VRP is the special case in which $|N_c| = 1$, i.e., the delivery location is fixed and does not change during the planning horizon.

2.2.1 Integer programming formulation

We next introduce an arc-based formulation of the VRPRDL as a mixed integer program. The formulation uses modeling techniques from routing problems with time windows, see, e.g. Desrochers, Lenstra, Savelsbergh, and Soumis [41]. We use the following decision variables:

$x_{ij} \in \{0,1\}$: indicates whether a vehicle travels from location i to j, for $i, j \in N \cup \{0\}$

$\tau_c \in [0,T]$: time of departure after service to customer $c \in C$ at any of its locations N_c

$y_c \in [0,Q]$: cargo remaining on vehicle after service to customer $c \in C$.

The formulation is then given by

$$\min_{x,y,\tau} \sum_{i,j \in N \cup \{0\}} w_{ij} x_{ij} \tag{2.2a}$$

$$\text{s.t.} \sum_{j \in N \cup \{0\} \setminus \{i\}} x_{ij} = \sum_{j \in N \cup \{0\} \setminus \{i\}} x_{ji}, \qquad \forall\, i \in N \cup \{0\} \tag{2.2b}$$

$$\sum_{i \in N_c} \sum_{j \in N \cup \{0\} \setminus \{i\}} x_{ij} = 1, \qquad \forall\, c \in C \tag{2.2c}$$

$$\sum_{i \in N} x_{0i} \leq |V|, \tag{2.2d}$$

$$\sum_{i \in N_c} a_i^c \sum_{j \in N \cup \{0\} \setminus \{i\}} x_{ij} \leq \tau_c \leq \sum_{i \in N_c} b_i^c \sum_{j \in N \cup \{0\} \setminus \{i\}} x_{ij}, \qquad \forall\, c \in C \tag{2.2e}$$

$$\tau_c + \sum_{i \in N_c} \sum_{j \in N_{c'}} t_{ij} x_{ij} \leq \tau_{c'} + T\left(1 - \sum_{i \in N_c} \sum_{j \in N_{c'}} x_{ij}\right), \qquad \forall\, c \in C \cup \{0\}, c' \in C \setminus \{c\} \tag{2.2f}$$

$$0 \leq y_c \leq Q - d_c, \qquad \forall\, c \in C \tag{2.2g}$$

$$y_c + Q\left(1 - \sum_{i \in N_c} \sum_{j \in N_{c'}} x_{ij}\right) \geq d_{c'} + y_{c'}, \qquad \forall\, c \in C \cup \{0\}, c' \in C \setminus \{c\} \tag{2.2h}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall\, i, j \in N \cup \{0\}. \tag{2.2i}$$

In the formulation we use the depot 0 both as a location and as a dummy customer, so that $N_0 = \{0\}$. Similarly, we can take $\tau_0 = 0$, $y_0 = Q$, and $[a_0, b_0] = [0, T]$. Constraint (2.2b) describes flow conservation for every location, and (2.2c) enforces exactly one visit per customer. Similarly, (2.2d) limits the total number of routes to the number of vehicles. Constraint (2.2e) enforces the time windows, while (2.2f) ensures that departure and travel times are consistent. Constraints (2.2h) and (2.2g) serve a similar function for the cargo variables.

2.3 Optimizing the delivery cost for a fixed customer sequence

The fact that a customer has multiple delivery locations during the planning horizon implies that specifying customer delivery sequences is no longer sufficient to specify a solution. In traditional vehicle routing settings, for a given sequence it is straightforward to ascertain that the delivery route is feasible and to determine an associated minimum cost. (In almost all VRP variants, for a given route it is optimal to deliver to a customer as early as possible.)

In the VRPRDL, a set of customer delivery sequences does not automatically provide a set of delivery routes, because a customer can be visited at one of several locations. Thus, a core optimization problem in the context of the VRPRDL is to determine optimal customer delivery times and locations for a given sequence. Throughout this section we assume a given customer sequence $(1, \ldots, m)$ with $d_1 + \cdots + d_m \leq Q$.

We solve the problem using forward dynamic programming (DP) on a time-expanded

network with nodes of the form (c, τ, i), where $c = 0, \ldots, m+1$ represents the current customer (and $0, m+1$ are, respectively, the start and end of the route), $\tau \in [0, T]$ is the departure time from customer c, and $i \in N_c$ is the location where the delivery to c took place; the latter can be inferred from τ and c's time windows, but we include it to simplify our exposition. The recursion is then given by

$$z(0, 0, 0) = 0 \tag{2.3a}$$

$$\begin{aligned} z(c, \tau, j) = \min\{z(c-1, \tau', i) + w_{ij} : \quad & i \in N_{c-1}, \quad a_i^{c-1} \le \tau' \le \min\{b_i^{c-1}, \tau - t_{ij}\}\}, \\ & c = 1, \ldots, m+1, \quad j \in N_c, \quad \tau \in [a_j^c, b_j^c]. \end{aligned} \tag{2.3b}$$

The quantity $z(c, \tau, j)$ is the minimum cost of a partial route $(0, \ldots, c)$ that delivers to customer c at location j at time τ, which implies that the minimum cost route is found by taking $\min\{z(m+1, \tau, 0) : \tau \in [0, T]\}$. The recursion (2.3b) states that the minimum cost of a delivery to customer c at location j at time τ is obtained as the sum of the minimum cost of a delivery to the previous customer in the sequence, i.e., $c-1$, at location i at time τ' in the time interval $[a_i^{c-1}, \min\{b_i^{c-1}, \tau - t_{ij}]$, i.e., the feasible departure times at location i that arrive at location j at or before time τ and the cost of travel from location i to location j. These quantities, as defined, range over all possible values of $\tau \in [0, T]$ at every step, which may be impractical even if all times are integer-valued, and may be intractable otherwise. However, we next show that the optimal route for the fixed sequence can be calculated efficiently.

Theorem 1. *For a fixed customer sequence $(1, \ldots, m)$, the optimal delivery route can be calculated in $O(k^2 m^2)$ time.*

Proof. To prove the theorem we must show that the number of nodes the algorithm explores does not grow excessively. Beginning recursion (2.3) at $(0, 0, 0)$, the algorithm only

evaluates nodes that are reachable from a previously evaluated node and non-dominated in terms of time and cost; the latter condition implies that the vehicle only waits at a location if it arrives before the time window.

We define a node to be *regular* if it is of the form (c, a_i^c, i), i.e. the vehicle delivers at the earliest moment in a location's time window, and *irregular* otherwise. From any node the algorithm is currently evaluating, (2.1b) and the triangle inequality guarantee that at most one of the non-dominated reachable nodes will be irregular: If location i is reachable within its time window, by condition (2.1b) the vehicle could "follow" the customer from there to any later location and make the delivery at the start of that window. Figure 2.1 shows an illustration of (the first layers of) a time-expanded network, with regular and irregular nodes.

The proof is completed by bounding the number of regular and irregular nodes the algorithm explores: Customer 1 has at most k nodes (with one possibly being irregular); these nodes then generate at most k irregular nodes for customer 2, which are added to its (no more than) k regular nodes. By induction, when customer c in the sequence is reached, no more than ck nodes will be evaluated, and thus we conclude that a total of $O(km^2)$ nodes and $O(k^2m^2)$ arcs will be evaluated by the algorithm. □

The proof of Theorem 1 implies that condition (2.1b) can be relaxed to require that the time between windows be at least the travel time between the corresponding locations. However, the argument breaks down if this time is shorter, because we can no longer guarantee that only one irregular node is generated at every evaluation.

The algorithm can be sped up heuristically by eliminating dominated irregular nodes. For any customer c and location i, if $z(c, \tau, i) \leq z(c, \tau', i)$ and $\tau < \tau'$, the latter node can be eliminated without evaluation, since any nodes it can reach are also reachable from the former at equal or lower cost.

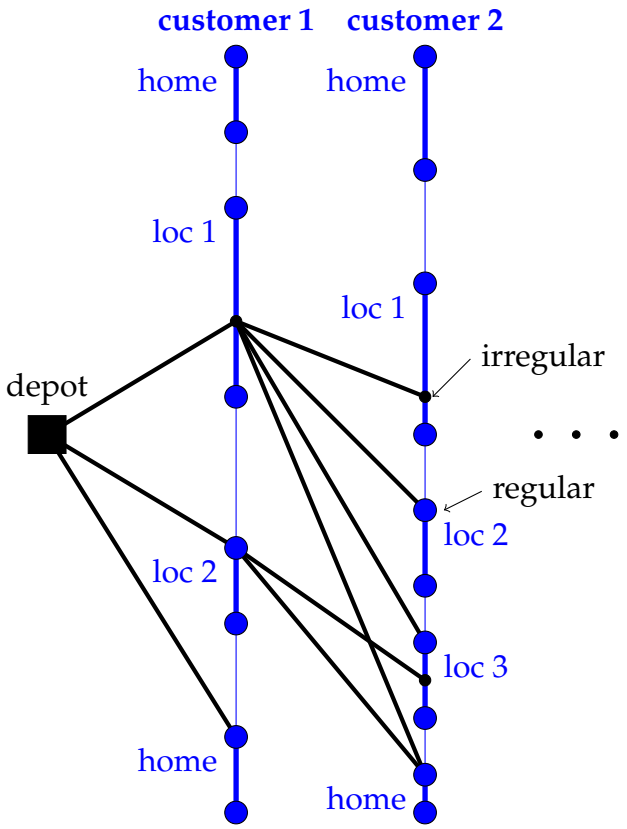


Figure 2.1: Illustration of the time-expanded network used in the dynamic program (showing the first two customers only).

2.4 Heuristics

The VRPRDL generalizes the VRP and is a difficult optimization problem from both a theoretical and practical perspective. As with the VRP, one option to produce high-quality solutions is via computationally efficient heuristic methods.

In the case of the VRPRDL, these methods can harness the flexibility afforded by the multiple delivery locations available per customer. Specifically, the heuristics we propose rely on applying many *enhanced* insertion and deletion operations to a given route; if we fix each customer's delivery location on the route, these operations are identical to their analogues in a classical VRP or other routing model with time windows. However, we can increase the operations' flexibility and the resulting potential of heuristics to produce high-quality solutions by considering also the shifting of customers to different locations (and thus a shift in the corresponding time windows) within the insertion and deletion procedure. To do so, given an incumbent route, in addition to the route itself and its time window feasibility information, we maintain a collection of alternate routes that differ from the incumbent in exactly one customer's delivery location; similar ideas have been used in other constrained vehicle routing contexts, e.g. [42].

Assume again that we have a fixed sequence of customers $(1, \ldots, m)$ with $d_1 + \cdots + d_m \leq Q$ and an incumbent feasible route $r = (i^1, \ldots, i^m)$ serving these customers in this order, where $i^c \in N_c$ for each $c = 1, \ldots, m$. To implement our enhanced insertion operations when time windows are present, it is necessary to calculate the "effective" time windows for each customer location, i.e. a window $[\alpha^c_{i^c}, \beta^c_{i^c}]$ specifying the earliest and latest times the customer can feasibly be served by a vehicle following this route. These windows are calculated using the recursions

$$\alpha^1_{i^1} = a^1_{i^1}, \qquad \alpha^c_{i^c} = \max\{a^c_{i^c}, \alpha^{c-1}_{i^{c-1}} + t_{i^{c-1},i^c}\}, \quad c = 2, \ldots, m; \tag{2.4a}$$

$$\beta^m_{i^m} = b^m_{i^m}, \qquad \beta^c_{i^c} = \min\{b^c_{i^c}, \beta^{c+1}_{i^{c+1}} - t_{i^c,i^{c+1}}\}, \quad c = 1,\dots,m-1. \tag{2.4b}$$

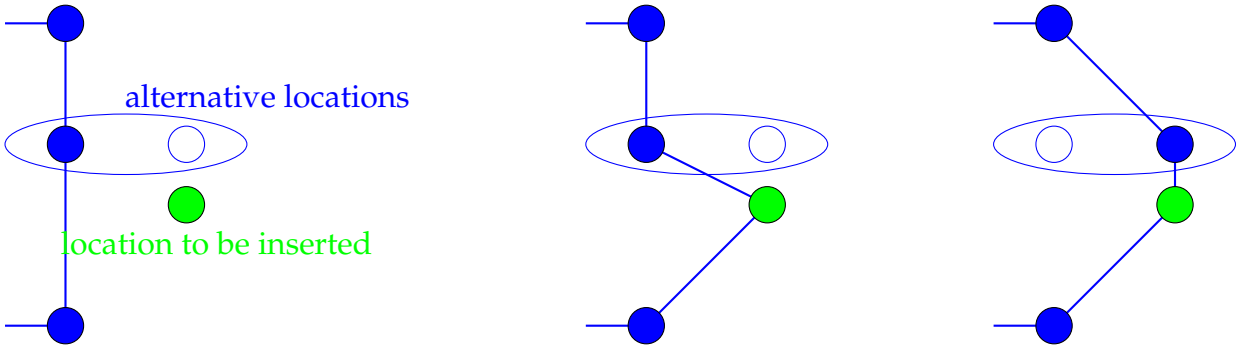
In addition, for each customer $c = 1,\dots,m$ we also maintain similar time windows for every location $i \in N_c \setminus \{i^c\}$ defined as

$$\alpha^c_i = \max\{a^c_i, \alpha^{c-1}_{i^{c-1}} + t_{i^{c-1},i}\}, \qquad \beta^c_i = \min\{b^c_i, \beta^{c+1}_{i^{c+1}} - t_{i,i^{c+1}}\}. \tag{2.4c}$$

Assuming $\alpha^c_i \le \beta^c_i$, this window represents the earliest and latest possible delivery to c if the delivery occurs at the alternate location i, with the remainder of the route unchanged. With these effective time windows available, we can verify whether a new customer $\hat{c}$ can be inserted at location $\hat{\imath} \in N_{\hat{c}}$ between customers $c-1$ and c while simultaneously switching $c-1$ to delivery location $i \in N_{c-1}$ and c to delivery location $j \in N_c$ if

$$\max\{a^{\hat{c}}_{\hat{\imath}}, \alpha^{c-1}_i + t_{i,\hat{\imath}}\} \le \min\{b^{\hat{c}}_{\hat{\imath}}, \beta^c_j - t_{\hat{\imath},j}\}; \tag{2.4d}$$

the check can be carried out in constant time. If the condition is satisfied, the two quantities become the new location's effective time window, and we redefine the α values for $c,\dots m$ and β values for $1,\dots,c-1$ accordingly in $O(km)$ time. The new route would have $\hat{c}$ in position c, with subsequent customers shifted forward by one position. This also means we can optimize the insertion of a customer at any position in the route, to any of the customer's delivery locations, while also changing its predecessor's and successor's locations in $O(k^3m)$ time. Similar but more restricted operations can reduce the complexity; for example, if the insertion changes only one of the predecessor's or successor's location, the operation only takes $O(k^2m)$ time. A similar but simpler check and update can be implemented to enhance the deletion of a customer from a route. Figure 2.2 and 2.3 illustrate the potential benefits of these enhanced insertion and deletion operations.

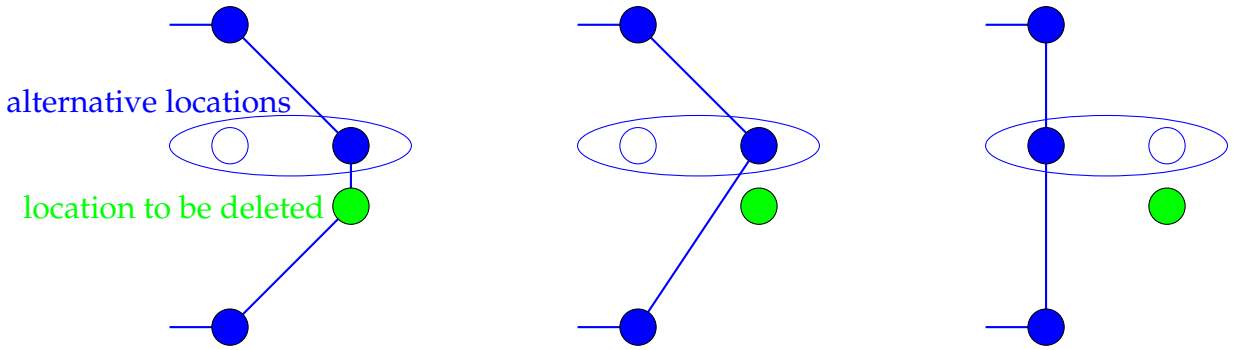


(a) Original route

(b) Standard insertion

(c) Enhanced insertion

Figure 2.2: Potential benefit of an enhanced insertion.



(a) Original route

(b) Standard deletion

(c) Enhanced deletion

Figure 2.3: Potential benefit of an enhanced deletion.

These basic (enhanced) insert and delete operations with simultaneous predecessor and/or successor location change can be embedded in more sophisticated heuristics and meta-heuristics. We discuss one possible pair of construction and improvement heuristics next.

2.4.1 Construction heuristic

The construction heuristic we propose for the VRPRDL is inspired by the family of *greedy randomized adaptive search procedures* [43]. The procedure repeats a randomized construction procedure N times, choosing the best solution. Each iteration produces a feasible solution by performing a basic insertion one customer at a time.

At each step in the construction, the next customer to be inserted is picked randomly from a restricted candidate list defined in the following manner:

1. For each customer c not yet assigned to a route in the solution, for each location $i \in N_c$, and for each route r among the routes in the solution (including the empty route), evaluate the cost change induced by the basic insertion operation at every point in the route (which may involve changing the predecessor or successor location).
2. Rank the options by nondecreasing cost and keep the K most profitable.

In our experiments, we used the parameters $N = 100$ and $K = 2$; we chose this construction heuristic with these parameters after comparing it in preliminary experiments against other methods, such as a simpler greedy rule and a carousel-greedy heuristic [44]. It also consistently outperformed the best solution found by Gurobi after two hours using the formulation (2.2). Algorithm 1 outlines the construction heuristic in pseudo-code.

Algorithm 1: CONSTRUCTION HEURISTIC

Input: Solution pool size N. Restricted candidate list size K.
Output: Solution S consisting of feasible routes serving all customers.

```
// Provided that the instance is feasible, this procedure
   returns a feasible solution with the least cost among N
   solutions constructed using a K-randomized greedy
   parallel insertion heuristic.

Pool ← ∅
for count ← 1 to N do
    R ← ∅                    // solution to be constructed
    Unserved ← C
    while Unserved ≠ ∅ do
        r, j, c, i ← RANDGREEDYPARALLELINSERTION(R, Unserved, K)
        // c to be visited at location i as j^th customer in route
           r
        if r = ∅ then
            return ∅    // unable to construct feasible solution
        else if r ∉ R then
            R ←append— r          // include new route in solution
        INSERT(r, j, c, i)
        Unserved ← Unserved\c
    Pool ←append— R
S ← arg min{cost(R) | R ∈ Pool}
return S
```

2.4.2 Improvement heuristic

The improvement heuristic we propose implements a variable neighborhood search using the destroy and recreate paradigm; see e.g. [45]. Each iteration of a destroy-recreate routine is composed of two phases. In the first phase, a feasible solution is "destroyed" by sequentially executing L basic delete operations. In the second phase, a new feasible solution is "recreated" through a sequence of basic insertion operations that reassign routes to the L deleted customers, respecting a first-deleted first-reinserted rule.

Different destroy-recreate neighborhoods can be induced by implementing different selection rules to determine the specific customer, location, and route on which an operation acts. We tested the following variants in our heuristic:

1. Greedy destroy and greedy recreate (GDGR).
2. Randomized greedy destroy and randomized greedy recreate (rGDrGR).
3. Random destroy and greedy recreate rule (RDGR).

At each step of a destruction phase, we consider a candidate deletion for every customer-route assignment in the (partial) solution. Among all these options, the greedy destroy rule executes the operation that yields the largest savings, whereas the K-randomized greedy destroy rule picks an option randomly among the K deletions with the largest savings. Finally, the random destroy rule chooses the customer to delete completely at random. Notice that the first and last rules are extremes of the randomized greedy rule.

We employ a similar rationale in the recreation phase, but the set of candidate insertions has a different form. First, because of the first-deleted first-reinserted rule, the customer c to reinsert is fixed. Second, c has no assigned route, position in the route, nor predetermined delivery location. Finally, inserting c into an "empty" new route may be the most appealing alternative. Thus, there is a candidate insertion for every delivery location i of customer c, for each route r in the solution (plus the empty route), and for

every position in the route.

In our implementation, to avoid local optima our heuristic alternates between rGDrGR and RDGR, switching whenever an improvement is not found after 10 iterations. During a switch, we employ a greedy destroy and then run the DP recursion (2.3) on each route in the partial solution before the next recreate phase. We run the procedure for a total of 2000 iterations, but after 1000 iterations we perform a one-time diversification, where the most expensive route is divided into two routes of approximately the same number of customers. We set the size of the restricted candidate list based on the number of customers, $K^{del} = \lceil 0.2n \rceil$ and $K^{ins} = \lceil 0.05n \rceil$. All of the parameter values were chosen based on initial tuning tests. Algorithm 2 details the improvement heuristic.

2.5 Computational study

The goal of our computational study is to demonstrate that (1) the algorithmic ideas presented form a solid foundation on which to build the enabling technologies to support trunk deliveries, and, more importantly, that (2) a trunk delivery system can have substantial economic and environmental benefits. To substantiate these claims, we present the results of a series of computational experiments. First, we demonstrate the effectiveness of our heuristics on a set of instances that were generated using as few assumptions as possible about customers' geographic profiles. Second, we focus on the value of trunk delivery systems by studying a set of instances generated to resemble practical situations. In all our experiments, total distance traveled is the relevant cost metric being minimized.

2.5.1 Instances

Two sets of randomly generated instances have been used in our computational experiment: a set of "general" instances and a set of "realistic" instances resembling situations likely to be encountered in real-life trunk delivery operations.

Algorithm 2: IMPROVEMENT HEURISTIC

Input:
S, a feasible solution
I, number of iterations
L, number of customers to be deleted and reinserted at each iteration
K^{del}, K^{ins}, size of restricted candidate lists
switch, number unsuccessful attempts before switching destroy and recreate neighborhoods
split, iteration at which the costliest route is broken into two
Output: S, a feasible solution

$lastImprovement \leftarrow 0$
$lastSwitch \leftarrow 0$
$L \leftarrow L^1$
for $iter \leftarrow 1$ **to** I **do**
  if $iter = split$ **then**
    $S \leftarrow$SPLITCOSTLIESTROUTE(S)
  $R \leftarrow$DESTROYRECREATE(S, L, K^{del}, K^{ins})
  if $cost(R) \geq cost(S)$ **then**
    if $iter - lastImprovement \geq switch$ **then**
      $R, Unserved \leftarrow$DESTROY($S, L, 1$)
      $R \leftarrow$DP(R)
      $S \leftarrow$RECREATE($R, Unserved, 1$)
      $lastSwitch \leftarrow iter$
      SWITCHNEIGHBORHOOD()
  else
    $S \leftarrow R$
    $lastImprovement \leftarrow iter$
return S

A general instance is characterized by a planning horizon, T, a number of customers, n, and for each of customer c, a demand, d_c, a home location, (x_c, y_c), a sequence of m_c roaming locations, in the order in which these locations are visited, and, finally, the time spent at every location (as a fraction of the planning horizon). The home location of each customer is reachable by an out-and-back tour from the depot, located at the center of the region, $(0, 0)$. The roaming locations of each customer are centered around the home location and can all be visited during the planning period in consecutive out-and-back trips from home (consequently, they can be visited in sequence, by the triangle inequality). The time *not* consumed by traveling is partitioned into $m_c + 1$ pieces of uniformly random lengths and linked to each location of the sequence (the home has two time windows, one at the beginning and one at the end of the planning period). A detailed description of the instance generator can be found in Algorithm 11 in the appendix. It has two important control parameters: the maximum number of roaming delivery locations visited by a customer and the maximum distance of a roaming delivery location from the home location. We have generated a set of 40 instances of increasing size and complexity with the number of customers ranging from 15 to 120, number of locations visited during the planning period ranging from 1 to 5 (if only one location is visited, it implies the customer stays at home the whole time), and a planning period of 12 hours.

A realistic instance, too, is characterized by a planning horizon, T, a number of customers, n, and for each customer c, a demand, d_c, and a home location, (x_c, y_c). However, the number of roaming locations is determined by the customer type, which can be one of three: at home only; at home and at work; and at home, at work, and somewhere else after work (e.g., at a shopping center, at a gym, or at a child's soccer practice). In all instances, the three types get 10%, 40%, and 50% of the customers, respectively. Work locations are in pre-defined clusters. Our instances are inspired by the geography of Atlanta and have eight work clusters. The coordinates of the centers of these work clusters (in minutes,

driving from downtown) can be found in Table 2.1.

Table 2.1: Coordinates of the centers of the work clusters.

Downtown	0	0
Buckhead	0	15
Airport	0	-15
Alpharetta	0	30
Marietta	-25	15
Doraville	25	15
Decatur	25	0
Douglasville	-25	0

The planning period is 14 hours long, extending from 6am to 8pm. Customers that work each have one of three schedule types: part-time, which implies the they work 4 hours starting either at 8am or at noon; almost full-time, which implies they work between 4 and 7 hours starting between 8 and 10am; and full-time, which implies they work between 7 and 8 hours starting between 8 and 9am. Everyone is assumed to arrive at work exactly at the time they start work. Of the customers that work, 10% are part-time, 10% almost full-time, and 80% are full-time. Customers that go somewhere else after work visit one of 30 locations spread across the region; specifically, they visit the after-work location that is closest to the home location. The time spent at the after-work location is randomly chosen, but is less than 50% of the time between the end of work and the end of the day. A detailed description of the instance generator can be found in Algorithm 12 in the appendix.

2.5.2 Algorithm performance

Because we are the first to explore the VRPRDL, there are no existing solution approaches to benchmark with, so we compare the quality of the schedules produced by our heuristic, which we denote by HEURRDL, to the quality of the schedules produced when we solve

the integer program (2.2) using the commercial integer programming software Gurobi 5.6 with a time limit of two hours. To ensure as fair a comparison as possible, we set the Gurobi search focus on feasibility (MIPFocus = 1) and provide the solution produced by our construction heuristic as a warm start. The parameter values used for the construction heuristic are $N = 100$ and $K = 2$, and the parameters used for the improvement heuristic are $I = 2000$, $K^{del} = \lceil 0.2|C| \rceil$, $K^{ins} = \lceil 0.05|C| \rceil$, $switch = 10$, and $split = 1000$.

The results for the 40 general instances can be found in Table 2.2, where we report the instance identifier, the number of customers in the instance, the total number of locations in the instance, the average percentage of time that a customer is available to receive deliveries (*i.e.* is not driving around), the cost of the schedule produced by the construction heuristic, c_{init}, the cost of the schedule after it has been improved, c_{heur}, the cost of the schedule produced by the integer programming solver, c_{IP}, and the relative difference in schedule cost, $(c_{IP} - c_{heur})/c_{heur}$ (as a percentage). The instances for which Gurobi was able to prove optimality are labeled with a check mark.

When provided with the initial solution produced by our construction heuristic, the integer programming solver was able to solve all instances with $n = 15$, all but one instance with $n = 20$, and two instances with $n = 30$ to optimality in two hours, but was unable to prove optimality for any of the instances with 60 or 120 customers. For the small instances ($n \leq 30$), HEURRDL produced schedules of comparable quality to those produced by the integer programming solver, and in all but one of the larger instances ($n \geq 60$) HEURRDL produced schedules of much better quality than the schedules produced by Gurobi; for the largest instances ($n = 120$) the cost is often reduced by more than 20%. We conducted a paired t-test on the difference of means of the solutions produced by HEURRDL and Gurobi for each group of instances of similar size, and found that the advantage in favor of HEURRDL is statistically significant at the 5% level for the groups of large instances and for the overall set of general instances.

Table 2.2: Performance analysis on general instances.

Instance	# Cust. (n)	# Loc.	Avg. % time available for delivery	c_{init}	c_{heur}	c_{IP}	$\frac{(c_{IP}-c_{heur})}{c_{heur}}$%
1	15	51	64.2	2463	2128	2128$^{\checkmark}$	0.0
2	15	53	69.9	2244	2007	1984$^{\checkmark}$	-1.1
3	15	53	71.3	4023	3661	3661$^{\checkmark}$	0.0
4	15	58	69.1	2746	2582	2572$^{\checkmark}$	-0.4
5	15	63	63.1	2101	1802	1802$^{\checkmark}$	0.0
			Group mean	2715.4	2436.0	2429.4	-0.3
6	20	64	69.5	3749	3374	3374$^{\checkmark}$	0.0
7	20	67	69.9	3042	2588	2588$^{\checkmark}$	0.0
8	20	69	67.7	2995	2489	2310	-7.2
9	20	77	53.9	2761	2536	2521$^{\checkmark}$	-0.6
10	20	81	60.0	3242	3196	2913$^{\checkmark}$	-8.9
			Group mean	3157.8	2836.6	2741.2	-3.4
11	30	76	78.1	4325	3659	3685	0.7
12	30	99	65.0	4975	4173	4173	0.0
13	30	104	64.2	4608	3849	3849	0.0
14	30	107	61.1	4617	3668	3659$^{\checkmark}$	-0.2
15	30	108	63.6	3102	2548	2543	-0.2
16	30	114	56.4	5382	4695	4656	-0.8
17	30	119	57.0	4315	3507	3492	-0.4
18	30	120	61.5	4317	3877	3875	-0.1
19	30	125	56.7	4082	3397	3390	-0.2
20	30	131	53.0	4842	3939	3936$^{\checkmark}$	-0.1
			Group mean	4456.5	3731.2	3725.8	-0.1
21	60	209	64.2	8370	6049	6121	1.2
22	60	214	58.9	7878	5873	6348	8.1
23	60	220	57.8	9843	8391	9029	7.6
24	60	226	61.0	9567	7670	7777	1.4
25	60	226	57.6	10354	9218	9093	-1.4
26	60	227	60.1	9425	8057	8058	0.0
27	60	230	63.9	9336	7032	7237	2.9
28	60	235	60.0	8260	6434	7607	18.2
29	60	236	56.0	10782	8971	10591	18.1
30	60	239	60.7	10048	8428	9853	16.9
			Group mean	9386.3	7612.3	8171.4	7.3*
31	120	423	62.9	17206	13414	16816	25.4
32	120	423	62.3	15367	11434	15244	33.3
33	120	429	59.7	16054	12987	15971	23.0
34	120	442	59.9	14408	11379	13434	18.1
35	120	452	56.6	14536	11713	14067	20.1
36	120	456	59.7	14323	11374	13596	19.5
37	120	462	60.5	13928	10516	13792	31.2
38	120	463	55.9	13455	11045	13312	20.5
39	120	468	56.0	13926	10115	13549	33.9
40	120	472	57.6	14005	10492	13892	32.4
			Group mean	14720.8	11446.9	14367.3	25.5*
			Overall Mean	7875.1	6356.7	7212.5	13.5*

$^{\checkmark}$ optimal value.
$*$ significant difference at the 0.05 level (paired t-test on c_{heur} and c_{IP} means).

In terms of the various components of the heuristic, including our problem-specific innovations, we observed the following behavior. The RDGR neighborhood (random deletions and greedy insertions) was far more effective in finding improvements than the rGDrGR neighborhood (randomized greedy deletions and randomized greedy insertions). In fact, for the larger instances ($n \geq 60$) no improved schedules were found using the rGDrGR neighborhood. On the other hand, for larger instances, the dynamic program, invoked during a switch of neighborhoods, often finds improvements. Furthermore, the flexibility to switch to an alternative delivery location is exploited regularly, especially on smaller instances ($n \leq 30$). The appendix includes detailed statistics on these experiments.

The instance characteristic most likely to impact the performance of an integer programming approach is the number of roaming delivery locations per customer. A second factor that may influence performance is the tightness of time windows (for which the average time available for delivery is a proxy). To quantify this impact, we next compare the quality of the solution produced by HEURRDL to the quality of the solution produced by Gurobi when the number of roaming delivery locations per customer changes. To this end, we construct a separate set of 25 general instances with 60 customers each, in which all customers have the same number m of roaming delivery locations for $m =$ 2, 3, 4, 5, and 6 (5 instances for each class). The results, summarized in Table 2.3, show that if customers have a large number of roaming delivery locations ($m \geq 5$), the quality of the solutions produced by HEURRDL is significantly better than the quality of the solutions produced by Gurobi. It appears that when the time available to make a delivery at customers decreases and the number of locations where such a delivery can be made increases, the linear programming relaxation becomes weaker and the integer programming solver struggles, while HEURRDL continues to find local improvements.

The correlation between number of locations and time available to receive deliveries

is quite strong ($\rho = -0.88$) in this sample, which means that a linear regression model including both predictors may be ill-conditioned. For this reason, we report the fit of the data onto a simple linear regression model, $\log(c_{IP}/c_{heur}) = \beta_0 + \beta_1 m + \varepsilon$: such model yields an adjusted coefficient of determination $Adj.R^2 = 0.47$, and regression estimates $\beta_0 = -0.064$ (p-value$= 0.002$), $\beta_1 = 0.020$ (p-value$= 0.0001$). A similar simple linear regression with time available for deliveries as predictor results in a model with much lower significance ($Adj.R^2 = 0.19$; slope coefficient -0.28, with p-value$= 0.017$), which supports our claim that, by far, the number of locations is the most important predictor of differences in performance between our heuristic and the IP solver.

Finally, we assess the performance of HEURRDL on the 40 realistic instances; the results can be found in Table 2.4. Again, HEURRDL does significantly better (in both the practical and statistical senses), as the cost of the solutions produced by Gurobi is on average 14% higher than the cost of the heuristic solutions.

2.5.3 The benefits of trunk delivery

To analyze the potential benefits of a trunk delivery system compared to a traditional home delivery system, we conducted computational experiments with both the general and the realistic instances.

First, we focus on the general instances, but use four different incarnations of each instance. In addition to the original instance, we consider three additional variations. In each of these variations, the locations visited by a customer during the planning period are relocated closer to the home location. Specifically, we take the line segment between a customer's home location and a location visited by the customer during the planning period and relocate that location on the line segment at distance $0.5d$, $0.25d$, and $0.125d$, respectively, where d is the original distance between the two locations. Furthermore, because the travel time between the locations is reduced, the time windows at the locations

Table 2.3: Performance analysis on general instances with controlled number of locations.

Instance	# Cust.	# Loc. per Cust.	Avg. % time available for delivery	c_{init}	c_{heur}	c_{IP}	$\frac{(c_{IP}-c_{heur})}{c_{heur}}$%
41	60	2	51.4	9178	8125	7877	-3.1
42			57.5	10650	9482	9378	-1.1
43			58.1	11754	10294	10423	1.3
44			63.7	8170	6541	6540	0.0
45			57.9	9088	8096	8079	-0.2
			Group mean	9768.0	8507.6	8459.4	-0.6
46		3	45.5	8652	7308	7264	-0.6
47			47.9	6368	5383	5038	-6.4
48			50.2	9325	7439	7326	-1.5
49			48.2	9807	8039	7670	-4.6
50			48.1	7570	6506	6187	-4.9
			Group mean	8344.4	6935.0	6697.0	-3.4*
51		4	41.6	7930	6561	6508	-0.8
52			46.9	8581	6959	7541	8.4
53			41.4	7928	6509	6426	-1.3
54			42.7	7319	6184	5981	-3.3
55			43.0	7799	6675	6024	-9.8
			Group mean	7911.4	6577.6	6496.0	-1.2
56		5	38.9	9711	8857	8602	-2.9
57			44.4	8455	6952	6980	0.4
58			39.9	7486	5865	7126	21.5
59			41.7	8918	7374	7867	6.7
60			40.8	8745	6953	8716	25.4
			Group mean	8663.0	7200.2	7858.2	9.1
61		6	38.3	8047	6567	7977	21.5
62			36.6	8228	6752	8123	20.3
63			40.3	8488	7262	7971	9.8
64			42.2	9760	8103	9630	18.8
65			39.0	7556	6285	7435	18.3
			Group mean	8415.8	6993.8	8227.2	17.6*
			Overall Mean	8620.52	7242.84	7547.56	4.2*

$^{\checkmark}$ optimal value.
$*$ significant difference at the 0.05 level (paired t-test on c_{heur} and c_{IP} means).

Table 2.4: Performance analysis on realistic instances.

Instance	# Cust.	# Loc.	Avg. % time available for delivery	c_{init}	c_{heur}	c_{IP}	$\frac{(c_{IP}-c_{heur})}{c_{heur}}$ %
r1a	60	200	89.4	514	443	487	9.9
r2a			88.9	579	511	553	8.2
r3a			89.0	723	597	694	16.2
r4a			89.3	578	467	527	12.8
r5a			88.5	545	478	537	12.3
r6a			88.0	604	490	604	23.3
r7a			88.4	579	466	579	24.2
r8a			89.1	541	413	444	7.5
r9a			88.2	628	499	530	6.2
r10a			88.9	521	450	512	13.8
			Group mean	581.2	481.4	546.7	13.6*
r11a	90	299	88.2	889	805	812	0.9
r12a			88.7	749	636	728	14.5
r13a			87.6	805	647	801	23.8
r14a			88.2	601	565	541	-4.2
r15a			88.5	759	659	759	15.2
r16a			88.4	780	685	777	13.4
r17a			89.2	830	717	802	11.9
r18a			88.0	683	592	638	7.8
r19a			88.2	822	687	802	16.7
r20a			88.3	709	602	659	9.5
			Group mean	762.7	659.5	731.9	11.0*
r21a	120	398	88.2	982	830	980	18.1
r22a			88.3	1025	855	1018	19.1
r23a			89.5	1008	903	995	10.2
r24a			88.1	1019	840	990	17.9
r25a			88.5	1029	875	968	10.6
r26a			88.2	1026	907	1009	11.2
r27a			88.3	983	864	982	13.7
r28a			88.6	1006	841	1006	19.6
r29a			88.7	979	815	966	18.5
r30a			87.9	936	750	909	21.2
			Group mean	999.3	848.0	982.3	15.8*
r31a	150	497	88.5	1078	932	1054	13.1
r32a			88.1	1154	949	1130	19.1
r33a			88.8	1205	976	1165	19.4
r34a			88.5	1042	878	1034	17.8
r35a			88.2	1082	962	1081	12.4
r36a			88.4	1099	973	1080	11.0
r37a			88.3	1138	905	1082	19.6
r38a			88.9	1025	938	977	4.2
r39a			88.1	1085	880	1067	21.3
r40a			88.8	1066	922	1066	15.6
			Group mean	1097.4	931.5	1073.6	15.3*
			Overall mean	860.15	730.1	833.625	14.2*

$^{\checkmark}$ optimal value.
$*$ significant difference at the 0.05 level (paired t-test on c_{heur} and c_{IP} means).

are increased.

For each of these four variants of an instance, we solve the VRPRDL using HEURRDL and compare the resulting schedule's cost to the cost of a home delivery (HD) schedule in which the customer remains at home throughout the planning horizon, solved with a simplified version of HEURRDL. We also consider a schedule that allows either home or roaming delivery (HRDL) to the customers, where delivery to the home can occur at any point in the planning horizon, but roaming deliveries must respect the time windows; this corresponds to the delivery company having the option to either deliver to the customer's car, or to leave the package at home. Table 2.5 outlines how these costs compare across instance variants. A few table entries are missing; for these instances there was no feasible solution to the VRPRDL. (By relocating the customer locations closer to the home location, an instance can become infeasible.)

We observe that the comparison between pure home and pure roaming delivery (ratio HD/RDL) is mixed, showing that either delivery system can outperform the other depending on the instance. On average, roaming delivery is 4 to 6% more expensive in the extreme cases (farthest and closest customer locations), and 4 to 5% less expensive in the middle cases. We conjecture the following explanations for this behavior. For the instance variants with locations closest to the customer's home location, the roaming locations are close enough to the home location that traveling to them is tantamount to visiting the home location, but the RDL instance is somewhat restricted by the time windows, while the HD location has no such restriction. Conversely, in the variants with locations farthest away from the customer's home location, the customer spends a significant amount of the planning horizon traveling between locations, and hence the delivery time windows are narrow and restrict the RDL instance's scheduling possibilities.

Since the HD and RDL systems exhibit somewhat complementary advantages, it is perhaps not surprising that the combination of the two delivery systems (HRDL) offers

Table 2.5: Comparison of roaming delivery to home delivery on general instances.

Instance	100% (d)			50% (0.5d)			25% (0.25d)			12.5% (0.125d)		
	HD / RDL	RDL / HRDL	HD / HRDL	HD / RDL	RDL / HRDL	HD / HRDL	HD / RDL	RDL / HRDL	HD / HRDL	HD / RDL	RDL / HRDL	HD / HRDL
1	0.917	1.159	1.063	1.003	1.066	1.069	1.055	1.000	1.055	0.917	1.127	1.033
2	1.302	1.012	1.317	1.255	1.007	1.264	1.082	1.000	1.082	1.317	0.798	1.052
3	1.187	1.071	1.271	1.041	1.022	1.064	0.921	1.123	1.034	1.186	0.853	1.012
4	0.919	1.214	1.117	1.065	1.026	1.092	1.056	1.000	1.056	0.927	1.116	1.034
5	0.959	1.126	1.079	0.983	1.092	1.074	1.019	1.017	1.037	0.959	1.063	1.019
Group geom. mean	1.046	1.114*	1.165*	1.065	1.042*	1.110*	1.025	1.027	1.053*	1.050	0.981	1.030*
6	0.856	1.446	1.238	1.044	1.056	1.102	1.039	1.024	1.063	0.858	1.198	1.028
7	0.963	1.176	1.133	0.942	1.116	1.051	1.021	0.992	1.013	0.963	1.055	1.017
8	0.929	1.251	1.163	1.092	1.000	1.092	1.051	0.985	1.036	0.981	1.021	1.002
9	0.959	1.272	1.220	1.071	1.037	1.111	1.060	1.002	1.062	0.964	1.074	1.035
10	1.135	1.070	1.215	1.045	1.110	1.160	1.053	1.000	1.053	1.113	0.944	1.051
Group geom. mean	0.964	1.237*	1.193*	1.037	1.063*	1.103*	1.045*	1.001	1.045*	0.972	1.055	1.026*
11	0.996	1.060	1.056	1.039	1.000	1.039	1.021	1.002	1.023	0.996	1.010	1.006
12	0.905	1.422	1.287	0.997	1.233	1.229	1.039	1.008	1.047	0.904	1.129	1.021
13	1.299	1.198	1.556	1.237	1.074	1.328	1.162	1.036	1.205	1.295	0.793	1.027
14	1.002	1.127	1.129	0.992	1.150	1.141	1.073	1.041	1.117	1.001	0.998	0.999
15	1.323	1.205	1.595	1.067	1.090	1.164	1.119	1.019	1.140	1.367	0.861	1.177
16	0.854	1.249	1.066	1.167	1.028	1.199	1.090	1.036	1.129	0.861	1.302	1.121
17	1.013	1.125	1.141	1.087	1.006	1.094	1.032	0.978	1.009	1.018	0.996	1.013
18	0.976	1.239	1.210	1.129	1.062	1.199	1.091	1.012	1.103	0.978	1.056	1.032
19	1.141	1.260	1.438	1.134	0.997	1.130	1.034	1.034	1.069	1.144	0.904	1.034
20	1.119	1.361	1.523	1.185	1.021	1.211	1.028	1.111	1.143	1.120	0.914	1.024
Group geom. mean	1.053	1.220*	1.285*	1.101*	1.064*	1.171*	1.068*	1.027*	1.097*	1.058	0.987	1.044*
21	1.143	1.193	1.364	-	-	1.188	1.038	1.083	1.124	1.151	0.894	1.029
22	0.885	1.297	1.148	1.079	1.103	1.189	1.061	1.041	1.105	0.929	1.119	1.039
23	0.833	1.481	1.234	1.040	1.197	1.245	1.072	1.074	1.151	0.840	1.223	1.028
24	0.796	1.395	1.111	1.078	1.068	1.151	1.038	1.006	1.045	0.812	1.268	1.029
25	0.795	1.613	1.282	-	-	1.253	1.103	1.026	1.132	0.821	1.354	1.112
26	1.003	1.380	1.384	-	-	1.106	1.045	1.068	1.116	1.032	1.058	1.092
27	0.836	1.361	1.138	-	-	1.058	0.993	1.043	1.035	0.837	1.216	1.018
28	0.826	1.343	1.109	0.928	1.237	1.148	1.085	1.067	1.157	0.885	1.249	1.105
29	0.887	1.256	1.114	0.926	1.166	1.080	0.964	1.079	1.040	0.890	1.194	1.063
30	0.935	1.426	1.334	1.000	1.248	1.248	1.040	0.996	1.036	0.971	1.059	1.028
Group geom. mean	0.888*	1.370*	1.217*	1.007	1.168*	1.165*	1.043*	1.048*	1.093*	0.911*	1.156*	1.054*
31	0.864	1.426	1.232	1.150	1.070	1.230	-	-	1.143	0.902	1.204	1.086
32	0.782	1.368	1.070	-	-	1.109	0.938	1.035	0.971	0.832	1.309	1.089
33	0.937	1.251	1.173	-	-	1.140	1.019	1.102	1.123	1.004	1.117	1.121
34	0.829	1.286	1.066	1.034	1.027	1.061	1.037	0.983	1.020	0.861	1.201	1.034
35	0.636	1.681	1.070	0.865	1.237	1.070	0.986	1.025	1.011	0.665	1.542	1.025
36	0.851	1.382	1.177	1.036	1.122	1.162	1.031	1.061	1.094	0.895	1.186	1.061
37	0.953	1.358	1.295	1.001	1.107	1.108	1.015	1.020	1.035	0.999	1.056	1.055
38	0.748	1.404	1.050	0.991	1.105	1.095	1.055	1.032	1.089	0.769	1.387	1.067
39	0.989	1.432	1.416	1.147	1.168	1.339	1.140	1.052	1.199	1.027	1.083	1.112
40	0.825	1.434	1.182	0.990	1.135	1.123	0.986	1.038	1.024	0.852	1.203	1.025
Group geom. mean	0.835*	1.398*	1.168*	1.023	1.120*	1.141*	1.022	1.038*	1.069*	0.874*	1.221*	1.067*
Overall geometric mean	0.94*	1.29*	1.21*	1.05*	1.09*	1.15*	1.04*	1.03*	1.08*	0.96*	1.09*	1.05*

$*$ significant difference at the 0.05 level (paired t-test using log of ratios).

significant benefits. The results indicate that this combination can significantly reduce delivery costs over home delivery regardless of whether RDL is more or less expensive, and the cost savings are proportional to how far locations are from the customer's home. For example, in the variants with locations farthest away, the savings are over 20% on average, even though RDL by itself is more expensive. Figures 2.4 and 2.5 show Instance 15 and its HD and HRDL solutions, respectively, in which the flexibility to deliver to the trunk of the car reduced the delivery cost by more than 50%. These results indicate that roaming delivery can significantly impact costs if deployed properly, perhaps in conjunction with home delivery instead of as a replacement.

Next, we focus on the realistic instances, using two different incarnations of each instance. In the original instance, the depot is located in the center of the region (Downtown Atlanta). In the variation, the depot is located in the southern part of the city. By comparing results for the two variants, we can investigate how sensitive these are to the location of the depot. The results can be found in Table 2.6 where, in addition to the cost ratios HD/RDL and HD/HRDL, we also report the cost ratio RDL/HRDL.

We observe that for the realistic instances the results are quite different than for the general instances. Trunk delivery, whether considered by itself or in combination with home delivery, offers significant cost reductions, more than 65%, on average, when the depot is in the center of the region, and around 40%, on average, when the depot is in the southern part of the city. As one illustration, Figure 2.6 shows the home delivery and roaming delivery solutions for realistic Instance 7 when the depot is in the center. Unsurprisingly, the cost reductions are smaller with a depot in the southern part of the region than with one in the center of the region: the potential for cost reductions when delivering to customers with a home location in the northern part of the region is, relatively speaking, smaller in the former case.

Interestingly, for these realistic instances, there is no noticeable difference between

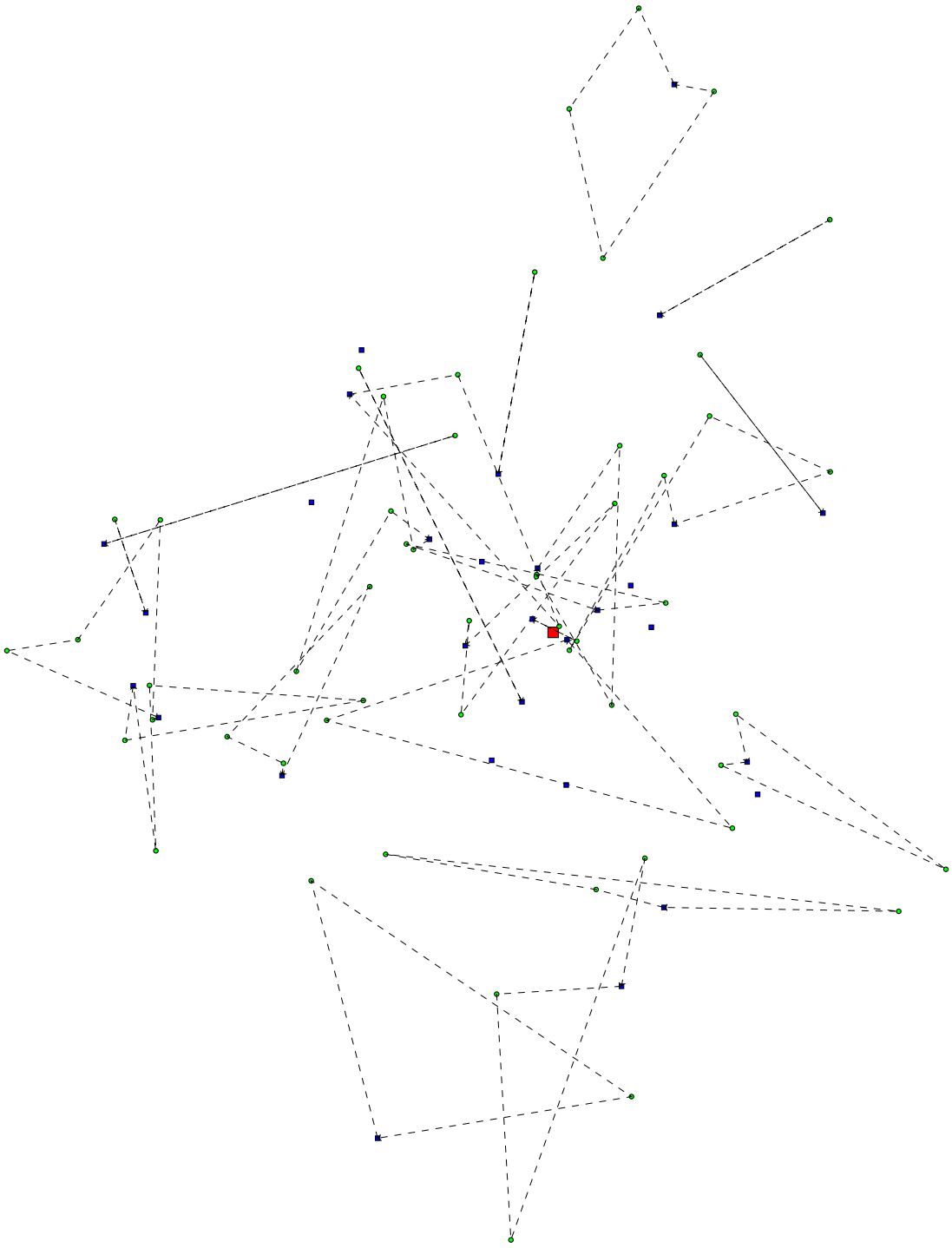
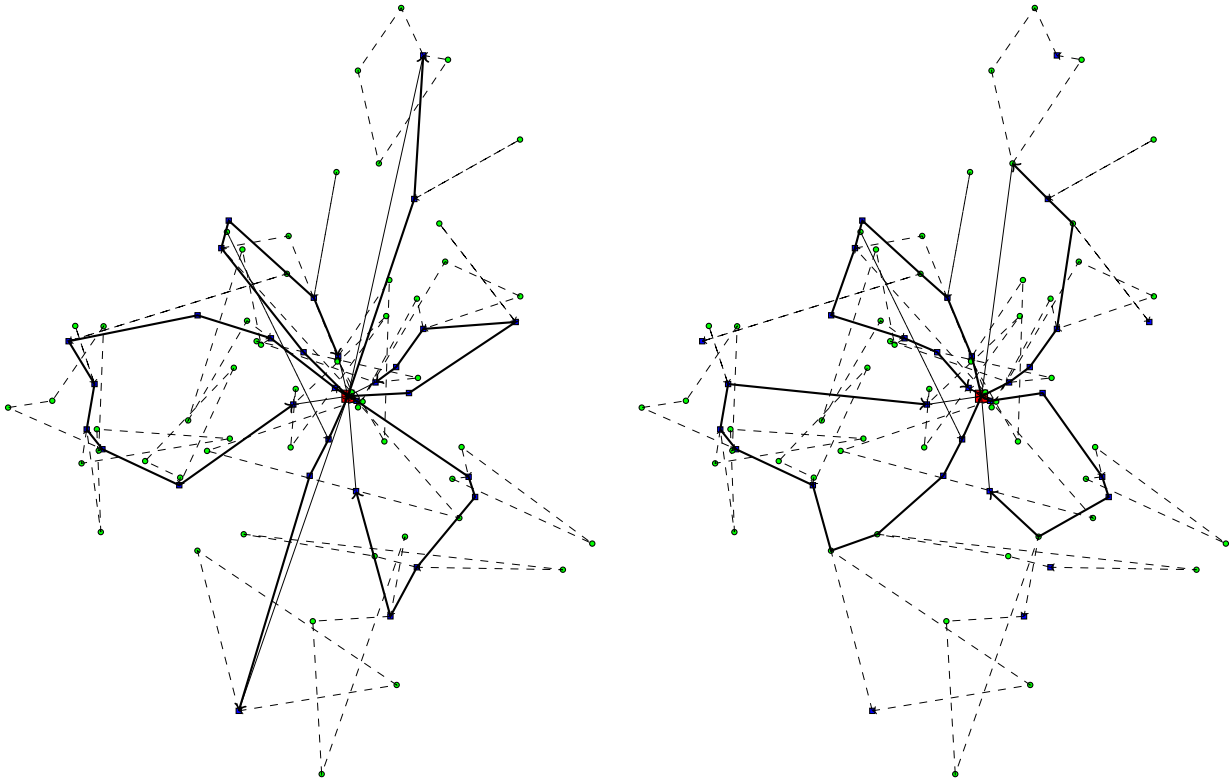


Figure 2.4: Instance 15 (depot - red square, home location - blue square, roaming location - green circle).



(a) Home delivery only

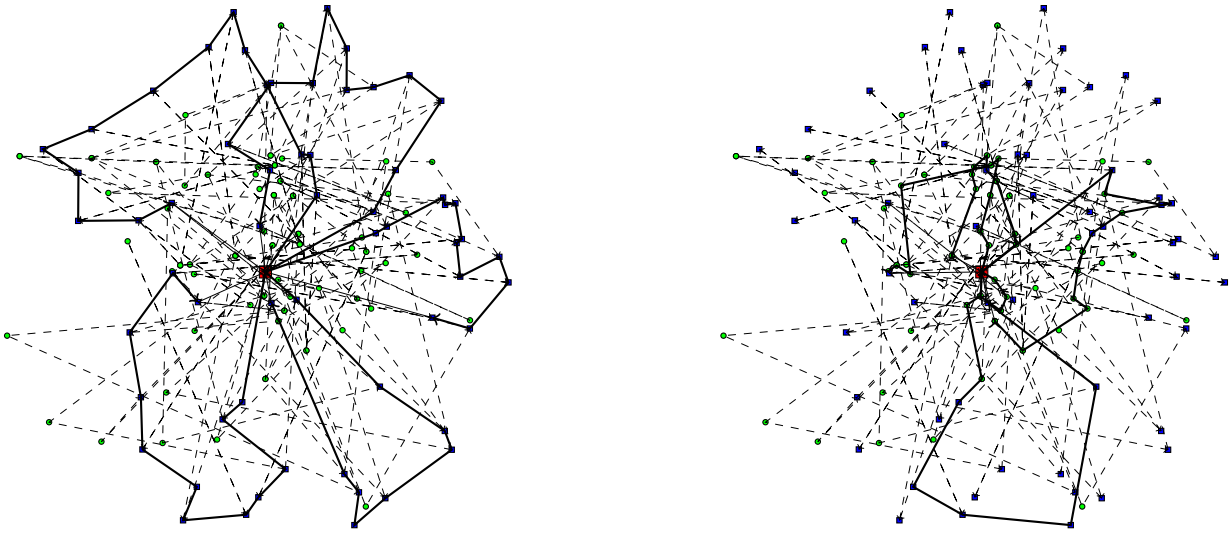
(b) Home and roaming delivery

Figure 2.5: HD and HRDL solutions for Instance 15 (routes - thick solid lines, not showing final return to depot).

Table 2.6: Comparison of roaming delivery to home delivery on realistic instances.

instance	Depot located at (0,0) HD/RDL	RDL/HRDL	HD/HRDL	Depot located at (0,-30) HD/RDL	RDL/HRDL	HD/HRDL
r1	1.901	1.021	1.940	1.596	0.991	1.582
r2	1.748	1.074	1.876	1.395	1.145	1.597
r3	1.454	1.099	1.599	1.382	1.037	1.433
r4	1.647	1.02	1.679	1.426	1.044	1.489
r5	1.766	0.976	1.722	1.410	1.018	1.436
r6	1.635	1.109	1.812	1.401	1.050	1.470
r7	1.745	1.131	1.973	1.537	1.009	1.550
r8	1.896	0.926	1.756	1.516	1.028	1.558
r9	1.521	1.018	1.549	1.371	0.985	1.351
r10	1.882	1.004	1.891	1.544	1.024	1.581
Group geom. mean	1.713*	1.036	1.774*	1.456*	1.032*	1.503*
r11	1.380	1.123	1.550	1.348	0.990	1.334
r12	1.711	0.991	1.695	1.396	1.029	1.436
r13	1.774	1.022	1.814	1.395	1.052	1.467
r14	1.858	1.046	1.944	1.519	1.047	1.591
r15	1.651	1.025	1.692	1.344	1.072	1.441
r16	1.587	1.018	1.615	1.361	1.033	1.407
r17	1.579	1.061	1.675	1.458	0.974	1.420
r18	1.779	0.988	1.758	1.470	0.974	1.431
r19	1.651	0.979	1.615	1.339	1.031	1.380
r20	1.832	1.067	1.956	1.517	0.995	1.509
Group geom. mean	1.674*	1.031*	1.727*	1.413*	1.019	1.440*
r21	1.558	0.995	1.550	1.298	1.106	1.435
r22	1.614	1.015	1.639	1.296	1.011	1.310
r23	1.411	0.993	1.402	1.284	1.053	1.352
r24	1.606	1.006	1.616	1.332	0.999	1.331
r25	1.544	0.994	1.535	1.283	1.061	1.361
r26	1.560	1.058	1.651	1.311	1.028	1.348
r27	1.514	1.026	1.553	1.211	1.078	1.305
r28	1.536	1.073	1.648	1.322	1.061	1.403
r29	1.656	0.967	1.601	1.360	1.002	1.362
r30	1.764	0.975	1.720	1.343	0.946	1.270
Group geom. mean	1.574*	1.010	1.589*	1.303*	1.034*	1.347*
r31	1.717	1.016	1.745	1.399	1.007	1.409
r32	1.663	0.992	1.649	1.289	1.095	1.411
r33	1.659	0.974	1.616	1.322	1.018	1.345
r34	1.858	1.038	1.928	1.541	0.947	1.460
r35	1.632	1.046	1.707	1.368	1.041	1.424
r36	1.646	1.010	1.664	1.434	0.973	1.396
r37	1.764	1.008	1.777	1.351	1.108	1.497
r38	1.667	1.004	1.675	1.350	1.019	1.375
r39	1.691	0.985	1.666	1.336	1.082	1.445
r40	1.709	0.997	1.704	1.441	0.985	1.420
Group geom. mean	1.699*	1.007	1.711*	1.381*	1.026	1.418*
Overall geometric mean	1.664*	1.021*	1.699*	1.387*	1.028*	1.426*

$*$ significant difference at the 0.05 level (paired t-test using log of ratios).



(a) Home delivery only

(b) Roaming delivery

Figure 2.6: HD and RDL solutions for Realistic Instance 7 (routes - thick solid lines, not showing final return to depot).

trunk delivery by itself or trunk delivery in combination with home delivery. This is in stark contrast to what we observed for the general instances, where trunk delivery in combination with home delivery clearly outperformed trunk delivery by itself. This may be explained by the clustering of work locations in the realistic instances, which offer significant opportunities to combine deliveries in very efficient routes; our results thus suggest that RDL may offer the most benefit in cities with a small number of work location clusters but more dispersed residential locations.

2.6 Final remarks

In this chapter, we introduced the VRPRDL as a canonical optimization model for the type of routing problem faced by companies deploying a trunk delivery system. The VRPRDL is a special case of the generalized VRP with time windows, in which the locations visited by a customer (more precisely the customer's car) define the clusters. The fact the time windows of the locations in a cluster reflect the travel itinerary of a customer

and are, therefore, non-overlapping, imposes significant (additional) structure. The VR-PRDL constitutes a challenging routing problem with important applications in last-mile delivery.

Our computational study suggests that trunk delivery offers a significant opportunity for last-mile package delivery companies to reduce the distance traveled and thus reduce both economic cost and emissions (and likely reduce congestion as well). Even for general instances that do not exhibit realistic geographic patterns, trunk delivery can significantly reduce distance traveled when compared to traditional home delivery. However, the true benefit of trunk delivery is even clearer in instances that reflect more realistic home and work geographical patterns; for these instances, our results suggest that trunk delivery alone could potentially reduce the distance traveled by 40% to 65%, depending on the location of the depot. These reductions in miles traveled would also have a significant environmental and social benefit.

Our results motivate a variety of questions. One example is whether it is possible to precisely characterize when a trunk delivery system is cheaper than a home delivery system and by how much: While our results lend credence to the intuition that trunk delivery can significantly outperform home delivery when many roaming delivery locations are clustered together (as in an office park area), there may be other important factors at play that can be uncovered through further statistical analysis.

In this initial assessment of the benefits of truck delivery, we have assumed that the travel itineraries of customers are known in advance and with complete certainty. Many companies, e.g. Roadie (`www.roadie.com`), are monitoring the travel behavior of individuals 24 hours a day and are developing machine learning technology to use the collected travel data to predict the daily travel itineraries of these individuals. However, predicted travel itineraries will not be perfect – for instance, an individual may leave work 30 minutes earlier than expected – and this suggests important avenues for future research,

including the planning of robust trunk delivery routes and dynamically adjusting trunk delivery routes based on real-time customer travel information.

Part II

Dynamic Delivery Systems

CHAPTER 3
A STUDY ON THE COMPLEXITY OF ROUTING PROBLEMS WITH RELEASE DATES AND DEADLINES

3.1 Introduction

Dynamic *delivery* problems are a class of dynamic vehicle routing problems, where vehicles deliver goods from an origin depot (or, perhaps a small number of origin depots) to customer locations, and the requests for delivery arise during the vehicle operating period; *e.g.*, [19]. When customer requests are revealed, they have both a deadline, specifying the latest time the delivery can be made, and a *release date*, specifying the earliest time the goods to be delivered will be ready to be dispatched from the depot. The release date can be the time that a request is made, but it can also be later to account for order picking and staging, preparation and assembly, etc.

What makes vehicle routing problems with release dates interesting and challenging is the trade-off between delaying the dispatch of a vehicle until more requests are ready, in order to build a route that serves many customers and has a low delivery cost per order, and dispatching a vehicle early, in order to have more time prior to deadlines for the vehicle to travel and deliver orders, *i.e.*, replacing nonproductive time at the depot waiting for orders to become ready with productive time on the road delivering orders. In dynamic settings, the challenge is even greater because of the uncertainty surrounding if and when future orders will be placed; delaying the dispatch of a vehicle (and, therefore, its return time) increases the risk of being unable to meet the deadline of future requests.

Archetti, Feillet, and Speranza [30] study the complexity of deterministic variants of some important classes of dynamic delivery problems. Specifically, they propose the

Traveling Salesman Problem with release dates (TSP-rd) in which a single vehicle operates one or more consecutive (non-overlapping in time) routes from a single depot, each time loading and delivering released orders to customer locations, and the Uncapacitated Vehicle Routing Problem with release dates (UVRP-rd) which differs only in that the routes may overlap in time (because multiple vehicles are available to execute the routes). For each problem, the authors consider two variants: in one, the objective is to minimize total distance traveled while completing all delivery routes by a common deadline T, and in the other the objective is to minimize the latest completion time of any route. Not surprisingly, all of these optimization problems are NP-hard when customer locations are nodes in a general network. However, the authors show that each of these problems is solvable in polynomial time for problem instances where customer locations and the depot are all points in the "real line" metric space $\mathbb{R}$, with distance and time between $x, y \in \mathbb{R}$ given by $d(x, y) = |y - x|$.

Specifically, Archetti, Feillet, and Speranza [30] develop dynamic programming algorithms for both variants of the TSP-rd problem that run in $O(n^3)$ time, and an $O(n^2)$ algorithm for UVRP-rd when minimizing distance traveled. In this chapter, we propose alternative dynamic programming algorithms with a complexity of $O(n^2)$ for the same two variants of the TSP-rd problem on the half-line. Furthermore, we extend the TSP-rd setting by considering *service guarantees*, which constrain each delivery to a customer to occur within a fixed amount of time S after the release date. We thus consider not only customer-specific release dates e_i, but also a customer-specific delivery deadlines $e_i + S$. Service guarantees are common in dynamic delivery contexts as retailers seek to offer instant gratification to consumers. For example, a retailer may guarantee online shoppers that an order will be delivered within two hours after the order is placed. Another environment where such a service guarantee arises naturally is meal delivery: "if your pizza is not delivered within 45 minutes after you place your order, it will be free." In addition

to the alternative algorithms for the variants considered in [30], we propose dynamic programs for TSP-rd problems with service guarantees that (i) minimize the completion time of the last route and (ii) minimize the distance traveled while completing the last route by deadline T, running in $O(n^2)$ and $O(n^3)$ time for instances on the half-line, respectively. We also develop an $O(n^3)$ time algorithm for the UVRP-rd problem with service guarantees on the half-line that minimizes distance traveled.

The remainder of the chapter is organized as follows. In Section 3.2, we formally introduce the problems studied and the notation used throughout. In Section 3.3, we prove a number of structural properties of optimal delivery schedules. In Section 3.4, we describe and analyze dynamic programming algorithms for single vehicle problems. In Section 3.5, we similarly describe and analyze a dynamic programming algorithm for the multiple vehicle problem minimizing travel distance. Section 3.6 contains some qualitative insights and concluding remarks.

3.2 A vehicle routing problem with release dates and order deadlines

In this section, we introduce an extension of the TSP-rd problem that includes order delivery deadlines. Let $N = \{1, \ldots, n\}$ be a set of customers located on the real half line $\mathbb{R}^+ \equiv [0, +\infty)$, and assume that the single depot is located at $x = 0$. Let the location of customer $i \in N$ be given by τ_i, and measure travel distances and times such that a round trip from the depot to i requires $2\tau_i$ distance and time. Note furthermore that a route traveling from the depot to customer i and back under these assumptions can also make deliveries to any set J of customers located such that $\tau_j \leq \tau_i$ for $j \in J$ while still incurring $2\tau_i$ distance and time.

Suppose each customer $i \in N$ places an order with a release time of r_i, which implies the earliest possible time a vehicle dispatched to deliver at i can depart the depot. In the remainder, we will use "customer" and "order" interchangeably for simplicity. Let

S be a common service guarantee applied to all orders, and therefore $r_i + S$ specifies the deadline time by which order i must be delivered at i.

Assume that deliveries to customers occur instantaneously upon vehicle arrival, and that all deliveries are made by the vehicle on the outbound journey from the depot. Thus, the latest possible dispatch time l_i from the depot such that the service guarantee at i is met is given by $l_i = r_i + S - \tau_i$. We restrict attention only to instances where $\tau_i \leq S$; instances not meeting this condition are trivially infeasible. Furthermore, we also restrict attention to instances where the latest time by which the final delivery route must be completed, T, is such that $T \geq r_i + S + \tau_i$ for all $i \in N$. This condition essentially states that the company will only accept a customer order such that if it is completed by its deadline, the vehicle has time to return to the depot on time. In problems without individual service guarantees, as considered in [30], the equivalent is to ensure that $T \geq r_i + 2\tau_i$ to avoid trivially infeasible instances.

Throughout this chapter, without loss of generality, we assume $r_i < r_{i+1}$ for $i < n$: since locations are on the half-line and delivery is instantaneous, if there were multiple orders released simultaneously, we need only consider the order furthest away from the depot (the rest of orders can always be delivered on time within the outbound journey of a feasible route to the furthest location).

The primary feasibility problem we study in this chapter is a single vehicle problem defined as follows:

Problem 1. *Is there a sequence of delivery routes that can be executed by a single driver, each starting and ending at the depot, such that each order $i \in N$ is dispatched at or after r_i and delivered at or before $r_i + S$, and the last route is completed at or before T?*

If customers $i_1, i_2, \ldots, i_k$ are served in delivery route K, then the earliest dispatch time of the route, $r(K)$, is $\max\{r_{i_1}, r_{i_2}, \ldots, r_{i_k}\}$, the latest dispatch time of the route, $l(K)$, is $\min\{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$, and the furthest order visited on the route, $\tau(K)$, is $\max\{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_k}\}$

(for convenience, the empty route has $r(\emptyset) = 0$, $\tau(\emptyset) = 0$, $l(\emptyset) = T$). Let the completion time of K, denoted by $c(K)$, be the time that the driver is back at the depot after serving orders in K. Observe that $r(K) \leq l(K)$ is necessary for feasibility, and that if the driver is at the depot at time $r(K)$, there is no reason to delay the dispatch of the route.

Before proceeding with the analysis of the problem, we introduce the notion of *non-interlacing* routes and delivery schedules that contain only non-interlacing routes. Let a route K be an ordered set of customers visited on a single dispatch from the depot.

Definition 1. Two routes K_1 and K_2 with $\min\{i \mid i \in K_1\} < \min\{j \mid j \in K_2\}$ are non-interlacing if and only if $\max\{r_i \mid i \in K_1\} < \min\{r_j \mid j \in K_2\}$.

Note that two routes serving a single customer each are always non-interlacing. In other words, any pair of interlacing routes involves at least three customers.

Definition 2. A delivery schedule $\mathcal{S}$ of non-interlacing routes is a partition of N that can be characterized by the set of last customers in each route, *i.e.*, $\mathcal{S} = \{i_1, i_2, \ldots, i_k, n\}$ with $1 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq n$, indicating that orders $\{1, \ldots, i_1\}$ are delivered on the first route, orders $\{i_1 + 1, \ldots, i_2\}$ are delivered on the second route, etc.

3.3 Structural properties of feasible and optimal delivery schedules

We now discuss key properties of feasible and optimal delivery schedules. First, observe that if K_1 and K_2 are non-interlacing routes with $r(K_1) < r(K_2)$, i.e., K_1 can be dispatched before K_2, then if K_1 and K_2 appear consecutively in an optimal delivery schedule, K_1 will be dispatched before K_2. Next, consider the following proposition which shows that we can limit our attention to delivery schedules for Problem 1 that contain only non-interlacing routes.

Proposition 2. *Any feasible delivery schedule for a single driver can be transformed into a feasible delivery schedule with non-interlacing routes, and no increase in total travel time.*

Proof. Suppose a feasible schedule contains interlacing routes $K_1 \supseteq \{i,k\}$ and $K_2 \supseteq \{j\}$, where $r_i < r_j < r_k$. Without loss of generality, suppose that (i,j,k) constitutes the earliest such order triplet, or, more precisely, suppose that (i,j,k) is the lexicographic minimum among the set of order triplets defining interlacing routes. This implies that i is the first order released in K_1, j is the first order released in K_2, and k is the first order in K_1 released after any order in K_2. Now consider the following cases:

1. Suppose K_1 is dispatched before K_2, which implies that $c(K_1) < c(K_2)$.
   - *If $\tau_j \leqslant \tau(K_1)$, then order j can be moved from route K_2 to route K_1 while preserving feasibility, thus creating new routes $K_1' = K_1 \cup \{j\}$ and $K_2' = K_2 \setminus \{j\}$.*

     Feasibility of K_1 and K_2 imply that a driver executing K_1 returns to the depot at time $c(K_1) \leqslant l(K_2) \leqslant l_j$, and the dispatch time of K_1 is $c(K_1) - 2\tau(K_1)$, with $r(K_1) \leqslant c(K_1) - 2\tau(K_1) \leqslant l(K_1)$. It follows that $r_j < r_k \leqslant r(K_1) \leqslant c(K_1) - 2\tau(K_1) \leqslant l_j - 2\tau(K_1) \leqslant l_j$, so K_1' can be dispatched at the same time as K_1. Furthermore, as $\tau_j \leqslant \tau(K_1)$, it will also return to the depot at the same time as K_1. Since $K_2' \subset K_2$, it follows that $c(K_1') = c(K_1) \leqslant l(K_2) \leqslant l(K_2')$. Finally, since $\tau(K_2') \leqslant \tau(K_2)$, a driver feasibly serving K_2' after K_1' can return to the depot no later than serving K_2 after K_1.
   - *If $\tau_j > \tau(K_1)$, then order k can be moved to route K_2, creating new feasible routes $K_1'' = K_1 \setminus \{k\}$ and $K_2'' = K_2 \cup \{k\}$.*

     If $c(K_1)$ is the time when a driver serving K_1 returns to the depot, feasibility of K_2 implies that $c(K_1) \leqslant l(K_2)$. And since $K_1'' \subset K_1$, a driver serving K_1'' can begin at the same time as when serving K_1 and return to the depot at time $c(K_1'') \leqslant c(K_1) \leqslant l(K_2)$. Furthermore, since $r_j < r_k$ and $\tau_j > \tau_k$, we have $l_k = r_k + S - \tau_k \geqslant r_j + S - \tau_k \geqslant r_j + S - \tau_j = l_j \geqslant l(K_2)$. It follows that $c(K_1'') \leqslant c(K_1) \leqslant l(K_2'') = l(K_2)$. Therefore, a driver can feasibly serve K_2'' after

K_1'' and return to the depot no later than serving K_2 after K_1.

2. Suppose K_2 is dispatched before K_1, which implies $c(K_2) < c(K_1)$.

   - *If $\tau_i \leqslant \tau(K_2)$, then order i can be moved from route K_1 to route K_2 while preserving feasibility, thus creating new routes $K_1' = K_1 \setminus \{i\}$ and $K_2' = K_2 \cup \{i\}$.*

     Since dispatching K_1 after $c(K_2)$ is feasible, it follows that $r_i < r(K_2) < c(K_2) \leqslant l(K_1) \leqslant l_i$. Hence, dispatching i together with K_2 at time $r(K_2)$ is feasible. Furthermore, $\tau_i \leqslant \tau(K_2)$ implies that $\tau(K_2') = \tau(K_2)$, and therefore $c(K_2') = c(K_2)$. Finally, because $K_1' \subset K_1$, it follows that K_1' can be served after K_2' and return to the depot at time $c(K_1') \leqslant c(K_1)$.

   - *If $\tau_i > \tau(K_2)$, then order j can be moved from route K_2 to route K_1, creating new feasible routes $K_1'' = K_1 \cup \{j\}$ and $K_2'' = K_2 \setminus \{j\}$.*

     Together, $r_i < r_j$ and $\tau_i > \tau(K_2) \geqslant \tau_j$ imply that $l(K_1) \leqslant l_i = r_i + S - \tau_i < r_j + S - \tau_j = l_j$, and therefore, $l(K_1'') = l(K_1)$. Furthermore, $K_2'' \subset K_2$, so if K_2'' is dispatched at the same time as K_2, K_2'' is guaranteed to complete at time $c(K_2'') \leqslant c(K_2)$. Also, $\tau(K_1) > \tau_i > \tau(K_2) > \tau_j$ implies that $\tau(K_1'') = \tau(K_1)$. In conclusion, K_1'' can be feasibly dispatched after K_2'' and complete no later than time $c(K_1)$.

Finally, observe that, after each application of one of these transformations, the first triplet of interlacing orders, (i', j', k'), is always lexicographically larger than the one in the previous schedule, (i, j, k). Since there is a finite number of orders, repeatedly applying these transformations on the earliest order triplet defining interlacing routes will eventually remove all interlacing routes without increasing the total travel time. □

Next, we observe that Problem 1 can be decided if the following optimization problem can be solved:

Problem 2. *Determine the minimum possible completion time, c^*, of a schedule $\mathcal{S}$ of delivery*

routes that can be executed by a single driver, each starting and ending at the depot, such that each order $i \in N$ is dispatched at or after r_i and delivered at or before $r_i + S$.

A dynamic programming algorithm that determines this minimum completion time will be presented in the next section. First, however, we conclude this section with a proposition that will ensure the correctness of our algorithm.

Proposition 3. *If a single-driver feasible delivery schedule minimizing the completion time exists, then there exists an optimal non-interlacing delivery schedule $\mathcal{S} = \{i_1, i_2, \ldots, i_k, n\}$ with the property that each partial schedule $\mathcal{S}_{[1,p]} = \{i_1, i_2 \ldots, i_p\}$, delivering orders $\{1, \ldots, i_p\}$, for $p = 1, \ldots, k$, completes the delivery of these order subsets as early as possible,* i.e., *$\mathcal{S}_{[1,p]}$ is an optimal delivery schedule for the (sub)problem defined by orders $\{1, 2, \ldots, i_p\}$.*

Proof. Let $\mathcal{Q} = \{q_1, q_2, \ldots, q_k, n\}$ be an optimal schedule, *i.e.*, a feasible schedule with minimum completion time $c^*(\mathcal{Q})$. Let $p \leq k$ be the largest index of a route in $\mathcal{Q}$ for which the partial schedule $\mathcal{Q}_{[1,p]}$ does not have minimum completion time for the (sub)problem defined by orders $\{1, \ldots, q_p\}$. Since $\mathcal{Q}_{[1,p]}$ is a feasible schedule for this (sub)problem, there must exist a schedule $\mathcal{Q}'_{[1,p]}$ with minimum completion time. We can replace Q by the schedule that concatenates $\mathcal{Q}'_{[1,p]}$ and $\mathcal{Q}_{[p+1,n]}$ (*i.e.*, the routes in $\mathcal{Q}$ delivering orders $\{q_p + 1, \ldots, n\}$) This concatenation is always feasible, since a driver completing all the routes consecutively in $\mathcal{Q}'_{[1,p]}$ can finish earlier than $\mathcal{Q}_{[1,p]}$, and thus be ready to serve the routes $\mathcal{Q}_{[p+1,n]}$. Repeatedly applying this transformation produces the desired delivery schedule. □

3.4 Dynamic programming algorithms for single vehicle problems

We now present optimal dynamic programming algorithms that solve TSP-rd problems with various constraints and objective functions.

3.4.1 Schedule Completion Time Problems

First, we consider problems with the objective to minimize the completion time of the delivery schedule. Before we consider our extended problem with service guarantees, we first develop an algorithm for the TSP-rd(time) problem in [30], which we restate:

Problem 3. *Determine the minimum possible completion time, c^*, of a schedule of delivery routes that can be executed by a single driver, each starting and ending at the depot, such that each order $i \in N$ is dispatched at or after r_i.*

When there is no service guarantee, we may assume, without loss of generality, that for $i < j$, we have $\tau_i \geqslant \tau_j$. This property follows from the fact that a dispatch at or after r_j covering order j must exist and must travel at least to τ_j, and will therefore serve all orders i' where $r_{i'} < r_j$ and $\tau_{i'} \leq \tau_j$ such that these orders can be ignored; see Archetti, Feillet, and Speranza [30] and Klapp, Erera, and A.Toriello [46].

Problem 3 can be solved by the following polynomial-time DP:

- states: i for $i \in \{0, 1, 2, \ldots, n\}$.
- values: $c(i)$ for $i \in \{1, 2, \ldots, n\}$ specifying the minimum completion time of a non-interlacing schedule serving orders $\{1, \ldots, i\}$.
- recursion:

$$c(0) = 0$$
$$c(i) = \min_{j \in \{0,\ldots,i-1\}} \left\{ \max\{c(j), r_i\} + 2\tau_{j+1} \right\}$$

We now prove the correctness of this algorithm. Consider the recursive step to compute $c(i)$. Order i is either included in a new last route together with orders $j+1, \ldots, i-1$, which is added to the partial schedule serving orders $\{1, \ldots, j\}$ when $j \leq i-2$, or in a

new route by itself when $j = i - 1$. In either case, the earliest possible completion time of the new schedule including i is the earliest dispatch time of this final route, $\max\{c(j), r_i\}$, added to the travel time $2\max_{k\in\{j+1,\ldots,i\}} \tau_k = 2\tau_{j+1}$. Because $c(n)$ is the minimum time required for a single driver to deliver orders $\{1, 2, \ldots, n\}$ and return to the depot after completing all deliveries, we can use this algorithm to decide a feasibility variant of Problem 3 by noting that a feasible delivery schedule exists if and only if $c(n) \leqslant T$.

Solving the forward recursion requires the evaluation of at most n possible actions at each of n stages, and evaluating each possible action takes a constant number of operations. Therefore, $c(n)$ is found in $O(n^2)$ time.

We now consider the problem with service guarantees, and develop an algorithm for Problem 2. Building on the insights above, a polynomial-time DP for this problem is given by:

- states: i for $i \in \{0, 1, 2, \ldots, n\}$.
- values: $c(i)$ for $i \in \{1, 2, \ldots, n\}$ indicating the minimum completion time of a non-interlacing schedule delivering orders $\{1, \ldots, i\}$, or ∞ if it is not possible to serve $\{1, 2, \ldots, i\}$ feasibly with a single driver.
- recursion:

$$c(0) = 0$$
$$c(i) = \min_{j\in\{0,\ldots,i-1\}} \left\{ \max\{c(j), r_i\} + 2\max_{j<k\leq i}\{\tau_k\} \;\middle|\; \max\{c(j), r_i\} \leq \min_{j<k\leq i}\{l_k\} \right\},$$

  where we stop the forward recursion and set $c(i) = c(i+1) = \ldots = c(n) = \infty$ if $c(i)$ is defined as the minimum over the empty set.

We now prove the correctness of this algorithm. Consider the recursive step to compute $c(i)$. Order i is either included in a new final route added to the partial schedule serving orders $\{1, \ldots, j\}$ when $j \leq i - 2$, or in a new route by itself when $j = i - 1$. In both cases,

the routes serving customers up to an including j remain feasible, and the new route is feasible if its dispatch time $\max\{c(j), r_i\}$ is not later than its earliest dispatch deadline $\min_{j<k\leq i}\{l_k\}$. The earliest possible return time is determined by adding the travel time of the new route $2\max_{j<k\leq i}\tau_k$ to the dispatch time.

We can compute all values $\max_{j<k\leq i}\{\tau_k\}$ and $\min_{j<k\leq i}\{l_k\}$ in advance in $O(n^2)$ time. After this, solving the forward recursion requires the evaluation of at most n possible actions at each of n stages, and evaluating each possible action requires a constant number of operations. Therefore, $c(n)$ is found in $O(n^2)$ time.

3.4.2 Schedule Travel Distance Problems

Next, we consider problems with the objective to minimize the total travel time of the delivery schedule. Again, before we consider our extended problem with service guarantees, we first develop an algorithm for the TSP-rd(distance) problem in [30], which we restate:

Problem 4. *Determine a sequence of delivery routes that can be executed by a single driver, each starting and ending at the depot, such that each order $i \in N$ is dispatched at or after r_i and that completes at or before time T with minimum total travel distance.*

As a first step, we use the DP from the prior section to verify that a feasible solution exists. Then, Problem 4 can be solved by the subsequently applying the following polynomial-time DP:

- states: i for $i \in \{1, 2, \ldots, n, n+1\}$.
- values: $v(i)$ for $i \in \{1, 2, \ldots, n\}$ indicating the latest dispatch time of a schedule of non-interlacing routes serving orders $\{i, \ldots, n\}$ that completes at or before time T.
- recursion:

$$v(n+1) = T$$

$$v(i) = \max_{j \in \{i+1,\ldots,n+1\}} \left\{ v(j) - 2\tau_i \mid v(j) - 2\tau_i \geq r_{j-1} \right\}$$

We now prove the correctness of this algorithm. Recall that without loss of generality in this case, $i < j$ implies $\tau_i \geqslant \tau_j$. It is feasible to serve order i together with orders $i+1, \ldots, j-1$, if the latest dispatch time $v(i)$ of this new first route is not less than r_{j-1}, and not greater than the latest dispatch time of the next route, $v(j)$, less the travel time $2\tau_i$ of the new route. Thus, we can set the new latest dispatch time to its largest value $v(j) - 2\tau_i$, but this is only feasible if this time is not less than r_{j-1}. The backward recursion ensures that the maximum dispatch time is selected among the feasible options.

Observe that the non-interlacing delivery schedule associated with time $v(1)$ does not contain any waiting time (otherwise there would be a non-interlacing delivery schedule that departs later) and thus has minimum total distance $T - v(1)$.

Solving the backward recursion requires the evaluation of at most n possible actions at each of n stages, and evaluating each possible action takes a constant number of operations. Therefore, $v(1)$ is found in $O(n^2)$ time after verifying feasibility with an initial algorithm that also requires $O(n^2)$ time.

Finally, we consider the problem of minimizing the total travel distance required by a delivery route schedule where visits to each customer meet a service guarantee.

Problem 5. *Determine a schedule of delivery routes that can be executed by a single driver, each starting and ending at the depot, such that each order $i \in N$ is dispatched at or after r_i and delivered at or before $r_i + S$ with minimum total travel distance.*

In this case, it again helps to construct a schedule backwards, but, because of the presence of service guarantees and associated latest dispatch times for orders, a delivery schedule that is pushed forward in time until it cannot be pushed forward anymore may still contain waiting time. As a consequence, it is no longer true that dispatching as late as

possible is equivalent to minimizing the total distance traveled. Therefore, the backward dynamic program developed below explicitly minimizes total distance traveled for *each possible time* that a delivery schedule may begin.

Since the dynamic program for this problem is slightly more complicated, we introduce some key notions first. Consider a value function $f_j(t)$ defined as the minimum distance required by a schedule that feasibly serves orders $j, \ldots, n$ by a driver available at the depot at time t; if no such feasible schedule exists, $f_j(t)$ will be set to ∞. Using this function, the optimal distance required to serve all orders is given by $f_1(0)$.

The dynamic program will rely also on value L_j for order j, defined as the latest time that a driver can depart the depot and feasibly serve the remaining orders $\{j, \ldots, n\}$ for $j = 1, \ldots, n$. To compute L_j, first define the following parameters associated with serving a group of orders $\{i, \ldots, j\}$ together on a single route from the depot: $\tau_{ij} = \max_{i \leqslant k \leqslant j} \tau_k$, the distance to the furthest delivery location, and $l_{ij} = \min_{i \leqslant k \leqslant j} l_k$, the latest possible dispatch time for the route. To compute values of L_j recursively, first let $L_{n+1} = T$. Then:

$$L_j = \max_{j \leq k \leq n} \left\{ \min\{l_{jk}, L_{k+1} - 2\tau_{jk}\} \;\middle|\; r_k \leqslant \min\{l_{jk}, L_{k+1} - 2\tau_{jk}\} \right\}$$

for $j = n, \ldots, 1$. Note that $\min\{l_{jk}, L_{k+1} - 2\tau_{jk}\}$ is the latest feasible departure time for the driver if he is to serve orders $j, \ldots, k$ together in a single trip before serving the remaining orders and r_k is the earliest feasible departure time if the driver is to serve orders $j, \ldots, k$ together.

The backward dynamic program for computing $f_j(t)$ is given below:

Initialization

$$f_{n+1}(t) = \begin{cases} 0 & \text{if } 0 \leqslant t \leqslant T \\ \infty & \text{otherwise} \end{cases}$$

Recursion

Given $j \in \{n, \ldots, 1\}$, compute for $k = j, \ldots, n$:

$$g_j^k(t) = \begin{cases} 2\tau_{jk} + f_{k+1}(\max\{r_k, t\} + 2\tau_{jk}) & \text{if } \max\{r_k, t\} \leqslant \min\{l_{jk}, L_{k+1} - 2\tau_{jk}\} \\ \infty & \text{otherwise} \end{cases}$$

Then

$$f_j(t) = \min_{j \leq k \leq n} \{g_j^k(t)\}$$

Note that $g_j^k(t)$ defines the minimum distance traveled by the driver if he departs at or after time t and on his first trip delivers orders $j, \ldots, k$ (or ∞, if to do so is infeasible).

Proposition 4. *The dynamic program correctly computes the value function $f_j(t)$ for $j = 1, \ldots, n$ and $t \in [0, T]$.*

Proof. We will argue that the dynamic program correctly computes the value function for (any partial) minimum distance non-interlacing schedule. Proposition 2 implies that this suffices.

We proceed by induction. Clearly, the dynamic program constructs an optimal value function when there is a single order. Next, assume that the dynamic program correctly computes the value function for any set of k or fewer orders. Given a set of $k+1$ orders $\{1, 2, \ldots, k, k+1\}$, the dynamic program computes the value function $f_1(t)$ for $t \in [0, T]$ as the point-wise minimum of a number of functions of the form

$$g_1^i(t) = 2\tau_{1i} + f_{i+1}(\max\{r_i, t\} + 2\tau_{1i}).$$

For each i, with $1 \leq i \leq k+1$, the function $g_1^i(t)$ represents the minimum travel distance of any schedule starting at t and serving orders $\{1, \ldots, i\}$ on the first delivery route. The travel distance of the first delivery route is $2\tau_{1i}$ and the earliest return to the depot is at

time $\max\{r_i, t\} + 2\tau_{1i}$. This partial schedule is extended with an optimal non-interlacing schedule serving orders $\{i+1, \ldots, k+1\}$, which adds $f_{i+1}(\max\{r_i, t\} + 2\tau_{1i})$ to the distance traveled. (By the induction hypothesis, the value function $f_{i+1}(\max\{r_i, t\} + 2\tau_{1i})$ yields the minimum travel distance for serving orders $\{i+1, \ldots, k+1\}$ as the set has at most k orders.) For a given t, this schedule is feasible, and therefore considered in the point-wise minimum operation, if and only if

a) the earliest departure time, $\max\{r_i, t\}$, of the first route delivering orders $\{1, \ldots, i\}$ is no later than l_{1i}, the latest possible departure that ensures that the service guarantee of the orders is satisfied, and

b) the earliest return to the depot, $\max\{r_i, t\} + 2\tau_{1i}$, of the first delivery route serving orders $\{1, \ldots, i\}$ is not too late for the timely delivery of orders $\{i+1, \ldots, k+1\}$ in subsequent routes, *i.e.*, no later than L_{i+1}.

From the previous observations it follows that, for any given t, if $g_1^i(t)$ is included in the point-wise minimum comparison defining $f_1(t)$, then $g_1^i(t)$ represents the extension of a non-dominated schedule serving orders $\{i+1, \ldots, k+1\}$ into a schedule serving all orders $\{1, \ldots, k+1\}$. Since for any given t, all feasible extensions from non-dominated schedules are considered, it follows that the best among them must be optimal. Therefore, for any $t \geqslant 0$, $f_1(t)$ represents the minimum total travel time of a non-interlacing schedule serving all orders $\{1, \ldots, k+1\}$ starting no earlier than time t, as conjectured. □

Proposition 5. *The value $f_1(0)$ can be computed in $O(n^3)$ time.*

Proof. First, we observe that any schedule that is feasible with a start at time t is also feasible with a start at time $s < t$. Therefore, the value function $f_j(t)$ is nondecreasing in t.

Next, we observe that $g_j^k(t)$ is a simple transformation of $f_{k+1}(t)$ involving a horizontal translation, with a "leveling-up" for values below a given threshold. To see this, we can

first compute

$$h_j^{k+1}(t) = \begin{cases} f_{k+1}(r_k + 2\tau_{jk}) & \text{if } 0 \leqslant t \leqslant r_k \\ f_{k+1}(t + 2\tau_{jk}) & \text{if } t > r_k \end{cases},$$

and then generate g_j^k via a vertical translation:

$$g_j^k(t) = h_j^{k+1}(t) + 2\tau_{jk}.$$

Since $f_n(t)$ is a nondecreasing step-function in t (until the dispatch deadline) and the operations outlined above preserve the functional form, we have that $f_j(t)$ is a nondecreasing step-function for $j = 1, \ldots, n$.

Furthermore, if s is a breakpoint of $f_j(t)$, then the optimal schedule that is dispatched at time s and serves orders $\{j, \ldots, n\}$ (with a first route delivering orders $\{j, \ldots, k^*\}$, and with total travel distance $g_j^{k^*}(s)$) cannot be started later. That is, if s is a breakpoint, then $s = \min\{l_{jk^*}, L_{k^*+1} - 2\tau_{jk^*}\}$, for some $k^* \geqslant j$. Thus, there can be no more than $n - j + 1$ breakpoints in $f_j(t)$.

Consequently, starting from the breakpoint description of $f_n(t)$, we can find a breakpoint description of $f_j(t)$ for $j = n-1, \ldots, 1$ (in that order). At each step, we first find the breakpoint descriptions of g_j^k for $k = j, \ldots, n$, which takes $O(n^2)$ time, and then determine the breakpoint description of $f_j(t)$ by comparing the functions $g_j^k(t)$, which also takes $O(n^2)$ time. Thus, computing the breakpoint description of $f_1(t)$ takes $O(n^3)$ time. □

3.5 A dynamic programming algorithm for the uncapacitated vehicle routing problem with release dates and order deadlines

The uncapacitated VRP with release dates (UVRP-rd) and order deadlines differs from the TSP-rd with order deadlines only in that the routes may overlap in time, since we assume that an infinite fleet of vehicles is available. As a consequence, a feasible solution

always exists, because each customer can be served on a route by itself; as noted also in Archetti, Feillet, and Speranza [30] for the case without order deadlines, such a solution is also optimal when the objective is to minimize the latest completion time of any route.

In this section, we develop an $O(n^3)$ algorithm for finding a set of routes serving all customers that minimizes the total distance traveled (to be executed by one or more vehicles). Our algorithm relies on the fact that each route in a solution can be executed by a different vehicle (because of the unbounded fleet), and on a critical fact analogous to Proposition 2, stated and proved next.

Proposition 6. *There exists an optimal solution to the uncapacitated vehicle routing problem with release dates and deadlines, in which all routes are non-interlacing.*

Proof. Suppose a feasible solution contains interlacing routes $K_1 \supseteq \{i, k\}$ and $K_2 \supseteq \{j\}$, where $r_i < r_j < r_k$. Without loss of generality, suppose that (i, j, k) constitutes the earliest such order triplet, or, more precisely, suppose that (i, j, k) is the lexicographic minimum among the set of order triplets defining interlacing routes. This implies that i is the first order released in K_1, j is the first order released in K_2, and k is the first order in K_1 released after any order in K_2.

Since we assume that K_1 and K_2 are delivered by different drivers, both routes can begin at their earliest dispatch times. It suffices to consider the following cases:

1. *Suppose $l(K_2) < r(K_1)$. Then order i can be moved from K_1 to K_2, thus creating new routes $K_1' = K_1 \setminus \{i\}$ and $K_2' = K_2 \cup \{i\}$.*

   Clearly, $r(K_1') \leqslant r(K_1) \leqslant l(K_1) \leqslant l(K_1')$ and $\tau(K_1') \leqslant \tau(K_1)$, so K_1' is feasible and does not incur more travel distance than K_1. Feasibility of K_1 and K_2 implies that $r_i < r(K_2) \leqslant l(K_2) < r(K_1) \leqslant l(K_1) \leqslant l_i$. Hence, $l(K_2') = l(K_2) \geqslant r(K_2) = r(K_2')$, so K_2' can be feasibly dispatched at time $r(K_2)$. Furthermore, if $h = \arg\min_{x \in K_2} \{r_x + S - \tau_x\}$, then $r_h + S - \tau_h = l(K_2) < r(K_1) \leqslant l_i = r_i + S - \tau_i$, and thus $r_h - r_i < \tau_h - \tau_i$.

Since $r_i < r_j \leqslant r_h$ (j is the first order released in K_2), it follows that $\tau_i < \tau(K_2)$. We conclude that K_2' incurs the same travel distance than K_2.

2. *Suppose* $l(K_2) \geqslant r(K_1)$. *Then,*
   - *if* $\tau_j \leqslant \tau(K_1)$, *order* j *can be moved to create feasible routes* $K_1' = K_1 \cup \{j\}$ *and* $K_2' = K_2 \setminus \{j\}$.

     Clearly, $r(K_2') \leqslant r(K_2) \leqslant l(K_2) \leqslant l(K_2')$ and $\tau(K_2') \leqslant \tau(K_2)$, so K_2' is feasible and does not incur more travel distance than K_2. Route K_1' can be feasibly dispatched because $l_j \geqslant l(K_2) \geqslant r(K_1) \geqslant r_k > r_j$, which means that $l(K_1') = \min\{l_j, l(K_1)\} \geqslant \max\{r(K_1), r_j\} = r(K_1')$. Furthermore, since $\tau_j \leqslant \tau(K_1)$, route K_1' does not require more travel than K_1.
   - *if* $\tau_j > \tau(K_1)$, *order* k *can be moved to create feasible routes* $K_1'' = K_1 \setminus \{k\}$ *and* $K_2'' = K_2 \cup \{k\}$.

     Clearly, $r(K_1'') \leqslant r(K_1) \leqslant l(K_1) \leqslant l(K_1'')$ and $\tau(K_1'') \leqslant \tau(K_1)$, so K_1'' is feasible and does not incur more travel distance than K_1. Furthermore, $r_k > r_j$ and $\tau_k \leqslant \tau(K_1) < \tau_j$ imply that $l_k = r_k + S - \tau_k > r_j + S - \tau_j = l_j \geqslant l(K_2)$, from which it follows that $l(K_2'') = l(K_2)$. Together with assumption $l(K_2) \geqslant r(K_1)$, this implies $l(K_2'') \geqslant \max\{r(K_2), r_k\} = r(K_2'')$, so route K_2'' is feasible. Furthermore, as $\tau_k < \tau(K_2)$, K_2'' does not require more travel than K_2.

Repeatedly applying these transformations in lexicographical order for all triplets defining interlacing routes, we can obtain a new feasible delivery schedule consisting only of non-interlacing routes without increasing the total distance traveled.

□

We are now in a position to describe our solution algorithm. First, observe that for any set of customers $\{k_1, k_1 + 1, \ldots, k_2\}$ we can quickly identify a furthest customer, say i ($k_1 \leq i \leq k_2$), and determine whether this set of customers can be visited feasibly by a route from the depot to i and back: the latest release date of any of the orders must be less

than or equal to the dispatch time of any of the orders, *i.e.*, $r_{k_2} \leq \min_{j=k_1,\ldots,k_2} l_j$ (where we use that $\max_{j=k_1,\ldots,k_2} r_j = r_{k_2}$). Since an unlimited vehicle fleet is available, such a feasible route can be dispatched exactly at time $\min_{j=k_1,\ldots,k_2} l_j$.

These observations allow us to define a shortest path problem on a digraph $D = (N, A)$ to find a minimum distance feasible solution. The node set N consists of nodes (k_1, k_2), representing a feasible route serving customers $\{k_1, \ldots, k_2\}$ ($k_1 \leq k_2$), a dummy source node s and dummy sink node t. The arc set A contains arcs of cost τ_i connecting nodes (k_1, k_2) and (k_1', k_2') with $k_1' = k_2 + 1$ and with i the furthest customer among $\{k_1, \ldots, k_2\}$. Arcs with a similar cost definition connect nodes (k, n) to t and, finally, zero-cost arcs connect s to nodes $(1, k)$. A set of routes visiting all customers and minimizing the total travel distance is obtained by finding a least-cost path from the source to the sink.

Note that the number of nodes in the graph is $O(n^2)$ (source, sink, and the combinations of k_1 and k_2). Furthermore, any arc into node (k_1, k_2) is of the form $((j, k_1 - 1), (k_1, k_2))$, which implies that the number of incoming arcs to any node is $O(n)$. This, in turn, implies that the maximum number of arcs in the graph is $O(n^3)$. Because the graph is directed and acyclic, a shortest path can be found in linear time on the number of arcs, *i.e.*, $O(n^3)$.

3.6 Discussion

By restricting attention to instances with delivery locations on the half-line, the sequencing of deliveries in a delivery route becomes trivial, allowing us to focus on which orders to serve together on a route, which relates to the trade-off mentioned in the introduction. We have been able to show that even in this simplified setting, and when assuming perfect information, deciding which orders to serve together on a route in the presence of release dates and deadlines induced by service-guarantees is nontrivial, but can be done

in polynomial time. We believe that the study of this simplified setting provides insights that are likely to be useful more broadly. For example, the structure of optimal solutions from our dynamic programs suggests that when seeking to minimize the return of drivers to the depot after their final delivery, a good heuristic strategy may be to always complete routes as early as possible, whereas when seeking to minimize the distance travel by the drivers, a good heuristic should try to plan routes that minimize the waiting time (given the available information).

While polynomial time algorithms most likely exist only for instances with very simple geometries, as in the case of delivery locations on a half-line, we highlight that for instances with delivery locations on a "star" network, *i.e.*, a central depot and a collection of m half-lines emanating from the depot, the structural property of optimal non-interlacing routes still holds, and our algorithms can be adapted to run in time $O(n^{(m+1)})$ for completion-time minimization, and $O(n^{(m+2)})$ for travel-time minimization. Furthermore, if there is an unlimited number of vehicles, the half-lines can treated independently. With this in mind, it may be possible to devise a cluster-first route-second heuristic for planar geometries to obtain perfect-information lower-bounds on the optimal objective value by pretending that customers in the same cluster are located on a half-line (carefully modifying the distance values).

A natural extension of this line of research is to re-introduce uncertainty into the model by considering instances with known delivery locations on a half-line, but uncertain release dates. When probability distributions of release dates are known, one can seek to minimize the expected completion time. Otherwise, one may consider and investigate online dispatching algorithms that attempt to balance the creation of cost-efficient routes and the preservation of future capacity.

CHAPTER 4
OPTIMIZATION ALGORITHMS FOR MEAL DELIVERY OPERATIONS

4.1 Introduction

As part of the broader trend of e-commerce proliferation, online restaurant aggregators – on-demand meal-ordering platforms where diners order their favorite cravings from an array of restaurants – are growing at a fast pace [20], and with them, the volume of meal delivery operations is rising quickly worldwide [47], increasing the potential for new economies of scope, scale, and density. According to Morgan Stanley, "online food delivery could grow by 16% annual compound rate in next 5 years"[48]. Aiming to capitalize on the market opportunity, emerging providers are investing heavily [49] on the deployment of meal delivery networks that promise restaurants and diners a reliable, fast and cost-effective delivery process.

While in the short-term the transition from restaurant-owned delivery services (which for many restaurants means no deliveries at all) to integrated meal delivery networks can be fueled by focusing on reliability and speed, the long-term sustainability of these networks is contingent on turning the efficiency potential into actual cost reductions. For this purpose, appropriate optimization technologies must be developed to solve increasingly large dynamic pickup and delivery problems in near-real time, and prescribe high-quality decisions able to control costs while satisfying very high service standards.

The successful deployment and operation of meal delivery networks is difficult not only due to the scale of these systems, but also due to the dynamism and urgency of arriving orders [25]. Without exaggeration, meal delivery is the ultimate challenge in last mile logistics: a typical order is expected to be delivered within an hour (much less if

possible), and within minutes of the food becoming ready, thus reducing consolidation opportunities and imposing the need for more vehicles operating simultaneously and executing shorter routes (which translates to an increased capacity need, in the form of courier hours). Furthermore, meal delivery networks must be able to respond to wide, and often abrupt, swings in demand both in spatial and time dimensions.

As mentioned in Chapter 1, to achieve the desired flexibility and responsiveness without incurring in significant fixed costs, meal delivery providers have adopted the "Uber" model of a "digital marketplace" of autonomous couriers. In exchange for internalizing risks and costs associated with demand uncertainty, independent contractors working as meal delivery couriers enjoy the relative freedoms to choose the hours they work (thus introducing uncertainty in scheduling), reject particular assignments without further justification (uncertainty in dispatching), and even decide whether to follow or disregard the suggested sequence of deliveries (uncertainty in routing).

In synthesis, meal delivery networks have ushered in dynamic pickup and delivery problems of unprecedented scale, and meal delivery providers are spearheading the adoption of flexible business models in the last-mile goods-transportation sector. In this chapter, we are concerned with laying out solid foundations for the design of optimization technologies that can scale up to the challenge, and we have made a methodological commitment to assume a dynamic deterministic framework, even if this means that novel and interesting questions (in particular, those related to courier autonomy) are not explored. The main contributions from the chapter are: i) a definition of the Meal Delivery Routing Problem (MDRP) to model the essential structure of this emerging class of dynamic delivery systems; ii) a solution algorithm based on a rolling-horizon repeated matching approach, which our computational results suggest to be robust and effective in realistic scenarios; iii) the release to the public domain of a set of realistic instances built from data provided by our industry partner, to facilitate benchmarking of alternative so-

lution methodologies; iv) an off-line decision support tool to determine a high-quality schedule of courier shifts (which has been used in the instance generation process).

We close this section with a brief survey of related literature, contextualizing some of the themes of later sections. In Section 4.2, we define the Meal Delivery Routing Problem, and then describe our solution algorithm (together with some possible variations) in Section 4.3. Next, in Section 4.5 we present and discuss the results of a computational study. Finally, in Section 4.6 we provide some concluding remarks.

4.1.1 Related literature

The MDRP belongs to the large family of dynamic vehicle routing problems (dVRP), more specifically, to the class of dynamic pickup and delivery problems (dPDP). A vast number of researchers have studied these problems from different angles for decades, and excellent surveys have been produced by Pillac, Gendreau, Guéret, and Medaglia [29] and Psaraftis, Wen, and Kontovas [22], in the case of dVRP, and Berbeglia, Cordeau, and Laporte [21], in the case of dPDP. In the next paragraphs we will limit ourselves to mention some recent studies that share some features with our work in terms of modeling or solution methodology.

Recent research in dPDP has focused mainly on the movement of people, *e.g.*, dial-a-ride and ride-sharing applications [50, 51], yet, despite the contextual difference, these problems share some important similarities with the MDRP: the difficulties posed by the increasingly large size of fully-dynamic instances, and the high urgency and low flexibility of the tasks to be scheduled. A frequently proposed strategy to deal with this challenge is the use of a myopic rolling horizon repeated matching approach, a scalable framework that has been shown to produce high quality solutions (*e.g.*, [52, 53]), making virtue out of necessity thanks to the low visibility of future events and the tight time constraints present in taxi and ride-sharing environments.

But perhaps the most natural label under which to catalogue the MDRP is that of dynamic *delivery* problems, a type of dPDP that has just recently begun to be studied on its own right, as same-day delivery services become more and more ubiquitous, in both simplified analytic systems (Archetti, Feillet, and Speranza [30], Klapp, Erera, and Toriello [54], and Reyes, Erera, and Savelsbergh [31]) and more realistic settings (Azi, Gendreau, and Potvin [19], Voccia, Campbell, and Thomas [55], Klapp [56], Archetti, Savelsbergh, and Speranza [57], and Dayarian and Savelsbergh [58]). In dynamic delivery problems, vehicles make multiple trips during the operating period to deliver goods locally from a depot (or a small number of depots, *i.e.*, restaurants, in the case of the MDRP) to customer locations. Due to the structure of the network (one or few depots), and also due to tight time constraints being modeled, dynamic delivery solutions have a special structure: once a vehicle is dispatched, modifying the route is highly undesirable or impractical. This structure is usually enforced directly in the model formulation, by requiring vehicles to be empty before starting additional pickup operations, which, while restrictive, is not at all unreasonable when dealing with deliveries of perishable goods, like meals. Ongoing assessments by Ulmer, Thomas, and Mattfeld [59] on the impact of relaxing this condition in the context of same day delivery suggest that allowing more flexible routes (preemptive returns in the single-depot case) may increase the number of customers served during the operating period, but the change on the average delivery times has not been thoroughly explored (and a trade-off, *i.e.*, an increase in delivery times, is plausible). Our proposed MDRP model definition does not impose that restriction, as in a multi-depot problem it is conceivable that routes may benefit from a vehicle making pickups while still executing deliveries from a *different* depot. However, the solution algorithm presented in this chapter always uses routes that finish all pending deliveries before a new pickup.

Having contextualized the problem, let us introduce a few more works which have informed our methods. The use of assignment models in rolling horizon strategies has

a long history in the realm of dynamic long-haul fleet management. Here we highlight the pivotal work by Yang, Jaillet, and Mahmassani [60], where the authors introduce a dynamic model that captures the essential characteristic of a real-time full truckload dispatching system, and compare a series of rolling horizon policies to assign (and schedule) jobs to trucks, under different operating conditions. The value of advanced information is measured by comparing the performance of myopic assignment policies and a policy that uses some *stochastic* knowledge about the expected location of future pickup and delivery points, and uses these expectations to account for the cost of moving trucks to serve future uncertain requests in an otherwise myopic algorithm. Beyond the obvious difference in time scales, the full truckload assumption is a significant departure from the meal delivery operating environment. However, we opt for defining route delivery segments (*i.e.*, consolidation decisions) before solving the assignment problem, which effectively reinstates a similar structure. While we do not use stochastic methods, we do explore the use of uncertain information about the future through the assignment of "follow-on" pick-up and delivery chains, where the later route segment will likely still change as more orders arrive, but already contains enough information to anticipate needed vehicle movements.

An issue of central importance in the design of dynamic algorithms is how to most effectively balance between the fulfillment of current tasks and the preservation of flexibility to complete future and unknown tasks. On this question, we highlight the work of Mitrovic-Minic, Krishnamurti, and Laporte [61], who propose a "double horizon" algorithm that evaluates the cost of actions (pickup or delivery insertions, in this case) using different cost functions if the actions occur in the short-term (within a given horizon) or in the long-term (anything beyond the horizon), an approach motivated by applications where time windows are wide and deliveries may occur much later than their corresponding pick-up operation (which is not the case for on-demand meal delivery). The double horizon heuristic is shown to outperform traditional (single) rolling-horizon al-

gorithms, on instances with time windows ranging between 1 hour and 8 hours from the release time, but with diminishing returns as the size of instances grows. Our solution algorithm incorporates a bi-objective mechanism with the same spirit, but operating in a different way (after all, the time constraints in meal delivery are very tight,*e.g.*, 40 minute target since announcement until delivery, unlike the problems for which "double-horizon" was conceived): we use information from an interval possibly different to the "assignment horizon" (*i.e.*, the window of orders to include in routes) to determine how intensely should consolidation opportunities be prioritized. At or before busy periods, like lunch and dinner time, when many orders arrive in a short time-span, the algorithm attempts to create more efficient routes, even at the expense of individual order service quality, while in relatively calm periods, the emphasis switches to delivering orders fast.

Another important question in dynamic routing is when to postpone and went to commit to the execution of decisions in order to mitigate uncertainty. On this topic, the work by Mitrovic-Minic and Laporte [62] comes to the fore. In [62], the distribution of waiting time for any given route is determined through one of 4 "waiting strategies": Drive First (DF: depart as early as possible), Wait First (WF: depart as late as possible), Dynamic Wait (DF within service zones, WF between service zones), and Advanced Dynamic Wait (DF within service zones, and slightly less "lazy" than WF between zones), where "service zones" are essentially clusters of consecutive stops in a route determined dynamically. In opposition to DF, WF tends to lead to more efficient routes, but at the risk of running out of vehicles to complete all deliveries by their deadline; as expected, the more complex strategy dominates the rest. In the context of meal delivery, service constraints are so restrictive that complex waiting strategies may not have a critical impact: it makes little sense for a vehicle to wait idle after delivering an order if there is more work to do, so the main question becomes how much to postpone departures from a restaurant. Our strategy in this regard is described in detail in Section 4.3.3, and it can be interpreted as a

restricted form of WF at restaurants.

Finally, the concepts of dynamism and urgency are fundamental for the study of dynamic delivery systems, where a large majority, if not the totality, of requests are revealed during the operating period (dynamic requests), and must be completed within a short time window (urgent requests). Naturally, the precise definition of these concepts has evolved throughout the years. Two decades ago, Lund, Madsen, and Rygaard [63], proposed to measure the *degree of dynamism* as the proportion of dynamic requests, a rough measure primarily designed for situations where at least part of the requests are "advance requests". To make meaningful comparisons between scenarios with any proportion of dynamic requests, Larsen, Madsen, and Solomon [64] later refined the definition and proposed the *effective degree of dynamism*, which attempted to capture both urgency and evolution of information in a single measure. This definition is the most popular one in recent literature. However, Lon, Ferrante, Turgut, Wenseleers, Berghe, and Holvoet [25] have recently argued that this popular measure has significant flaws, and that dynamism and urgency should be measured separately to correct this. After defining two independent measures, they empirically observe that dynamism, urgency and "cost" are related in a non-linear way: low dynamism and high urgency lead to higher costs, but high dynamism and high urgency do not; higher urgency leads to higher costs; and that cost is largely insensitive to changes in dynamism, all else equal (except for high urgency scenarios, as previously noted). This is consistent with the observations of Lund, Madsen, and Rygaard [63], who noted that "even with a large number of dynamic requests, it is possible to produce good solutions, provided that the dispatcher receives the requests way ahead of the actual service time." In this chapter, we adopt the definitions of [25].

4.2 The meal delivery routing problem

In this section, we introduce a stylized model of meal delivery operations, with the goal of formalizing what we consider to be their main structural features, from observation and interviews with our industry partner: multiple pick-up points (restaurants), fully dynamic order arrivals, fleet of vehicles with capacity distributed throughout the day in the form of shifts, the possibility to pick up multiple orders simultaneously, among others. Being a first attempt to study such systems, we have assumed a deterministic dynamic framework in our modeling, and, in our opinion, the most important feature left out is the ability of couriers to turn down specific requests, and to relocate freely when idling. Having said this, let us introduce the model:

Let R be a set of restaurants, and let each restaurant $r \in R$ have an associated location ℓ_r. Let O be a set of orders, and let each order $o \in O$ have an associated restaurant $r_o \in R$, a placement time a_o, a ready time (*i.e.* a release date at the restaurant) e_o, and an order drop-off location ℓ_o. Let C be a set of couriers, where each courier $c \in C$ has an on-time e_c (when the courier goes on duty), an on-location ℓ_c (where the courier will be at time e_c), and an off-time $l_c > e_c$ (when the courier goes off duty). Assume that all information about R and C is known *a priori*, but information about any particular order $o \in O$ is revealed only at its placement time a_o. The meal delivery routing problem consists of determining feasible routes for couriers to complete the pick-up and delivery of orders, with the objective to optimize a single or multiple performance measures. The structural assumptions that determine feasibility conditions, as well as relevant performance criteria are defined as follows.

4.2.1 Structural assumptions

Orders from the same restaurant may be combined into "bundles" with multiple drop-off locations, where the ready time of a bundle is the latest ready time of the constituent orders. There is no limit to the number of orders that may be combined into a bundle.

The travel time between any pair of locations is assumed to be invariant over time. The service time associated with the pickup of an order at a restaurant, s^r, represents the time a courier needs to park his vehicle, walk to the restaurant, pick up one or more orders, and walk back to his vehicle. The value of s^r is invariant over time and independent of the number of orders being picked up. Similarly, s^o, represents the service time associated with the delivery of an order at a customer location, i.e., the time a courier needs to park his vehicle, walk to the customer, drop off an order, and walk back to his vehicle.

The pickup time of an order (or bundle) at a restaurant is not smaller than the maximum of a) the order ready time and b) the courier arrival time to the restaurant plus half of the service time at a restaurant. The departure time from a restaurant is the pickup time plus half of the service time. The drop-off time of an order is the arrival time of the courier at the customer location plus half of the service time at a customer location. The departure time after delivering an order is the drop-off time plus half of the service time.

Couriers cannot pick up any orders after their off-time, but are allowed to drop off orders after their off-time. More importantly, in this deterministic model, it is assumed that couriers do not execute any autonomous decisions while on duty. In particular, they always accept any instruction handed to them, and they always wait for (new) instructions at their on-location and at the last location of their active assignment. Furthermore, couriers cannot receive instructions while executing an assignment and, thus, cannot be diverted while en-route to a restaurant.

Payments to couriers follow a guaranteed minimum compensation scheme: a courier

earns p_1 per delivered order, or is compensated at the rate of p_2 per hour, whichever is higher. Thus a courier collects $\max\{p_1 n,\ p_2(l_c - e_c)\}$, where n is the total number of orders served by the courier.

We consider the following variations in the operating environment:

- *Prepositioning*. Without prepositioning, the only instruction that can be sent to a courier is to pick up and deliver an order (possibly a bundled-order). With prepositioning, either the instruction to move to a restaurant (a prepositioning move) or the instruction to pick up and deliver an order can be sent to a courier.
- *Assignment updates*. Without assignment updates, once the instruction to pick up and deliver an order (possibly a bundled-order) has been communicated to a courier, this assignment has to be completed before a courier can receive a new instruction. With assignment updates, when a courier arrives at a restaurant to pick up an order (possibly a bundled-order), an instruction can be sent to the courier updating the order to be picked up. For example, the initial instruction may have indicated that order o_1 had to be picked up and delivered, and the update instruction may indicate that order o_1 and order o_2 have to be picked up and delivered (a bundled-order). Note that the only assignment update allowed is an update to the set of orders picked up at a restaurant.

4.2.2 Performance metrics

Given that meal delivery providers must bring together diners, restaurants and couriers, each with their own concerns, a number of performance measures are of interest. Some, like click-to-door, which is the difference between the drop-off time of an order and the placement time of an order, involve a target value, τ, and a maximum allowed value, $\tau_{\max}$. The primary performance measures for the meal delivery routing problem are:

1. Number of orders delivered.
2. Total courier compensation.
3. Cost per order: total courier compensation divided by number of orders delivered.
4. Fraction of couriers receiving guaranteed minimum compensation.
5. Click-to-door time: the difference between the drop-off time and the placement time of an order.
6. Click-to-door time overage: the difference between the drop-off time of an order and the placement time of an order plus the target click-to-door time.
7. Ready-to-door time: the difference between the drop-off time of an order and the ready time of an order.
8. Ready-to-pickup time: the difference between the pickup time of an order and its ready time.
9. Courier utilization: the fraction of the courier duty time that is devoted to driving, pickup service, and drop-off service (as opposed to time spent waiting).
10. Courier delivery earnings: courier earnings when considering only the number of orders served.
11. Courier compensation: the maximum of the guaranteed minimum compensation (based on the length of the duty period) and the delivery earnings.
12. Orders delivered per hour: for each courier, the number of orders delivered divided by the length of the shift.
13. Bundles picked up per hour: for each courier, the number of order bundles assigned divided by the length of the shift.
14. Orders per bundle: for each assignment, the number of orders to be picked up.

Relevant summary statistics for measures 5-13 are: average, standard deviation, minimum, 10-th percentile, median, 90-th percentile, and maximum.

4.3 A rolling horizon algorithm for the MDRP

In light of the highly dynamic and urgent nature of meal delivery operations, making a detailed plan to serve orders that are not ready in the near future, based only on deterministic information, is unlikely to be of much advantage. For this reason, in this chapter we propose to solve the MDRP using a rolling horizon matching-based algorithm that every f minutes uses a matching problem to prescribe only the next pick-up and delivery assignment for each courier. Individual orders may be bundled to be picked-up together and then delivered by a single courier following a specified route. After defining tentative courier - order assignments, a "commitment strategy" dictates which of these decisions are postponed and which are communicated to couriers.

Furthermore, since a bundle cannot be picked up before all orders are ready, the assignment of a bundle with an order ready far into the future is likely to be postponed (regardless of the ready times of the rest of orders in the bundle), our algorithm focuses on finding assignments only for the subset of known upcoming orders whose ready time falls within $t + \Delta_u$, the *assignment horizon*, $U_t = \{o \in O_t : e_o \leq t + \Delta_u\}$. So, at optimization time, t, the algorithm determines the best assignment of orders in U_t to the couriers on duty.

4.3.1 Bundles and routes

While ideally every order should be delivered by its target delivery time, this goal must be reconciled with the reality of a limited pool of couriers. Especially during busy periods of the day, it may not always be possible to pick up orders as they leave the kitchen and deliver them individually. Therefore, to better utilize capacity, couriers may pick-up and deliver multiple orders, increasing the utilization of couriers at the expense of some freshness loss. At optimization time t, the algorithm determines how many orders should

be in a bundle (defining a target bundle size), and then defines a satisfactory grouping and sequencing of the individual orders that will be assigned to couriers. Since we always define bundles with a unique route associated to them, from now on we may refer to bundles as routes indiscriminately.

System intensity and a target bundle size. A target bundle size may simply be defined as a fixed number throughout the whole operating period, or throughout predefined intervals, such as "lunch" and "dinner" times. However, to induce robustness and responsiveness in our solution method, we define such target in a dynamic fashion, in direct relation to a fraction of the form $^{\text{(\#orders ready)}}/_{\text{(\# couriers available)}}$, a rough measure of the amount of work that must be completed with the available resources during a given period of time. By doing this, we intend to encourage quick deliveries when there are fewer orders than couriers, and favor larger bundles when there are more orders than couriers and the system is under pressure. A parametric definition of the target bundle size at optimization time t is:

$$Z_t = \left\lceil \frac{|\{o \in O_t : \ e_o \leq t + \Delta_1\}|}{|\{d \in D_t : \ e_d \leq t + \Delta_2\}|} \right\rceil, \ \ \Delta_1 > 0, \Delta_2 > 0$$

where e_o is the ready time of order o and e_d is the time when courier d becomes available for a new assignment. Note that it is possible that there are no couriers available before $t + \Delta_2$, in which case Z_t is set to some default value. Specific values for Δ_u, Δ_1 and Δ_2 are set through a tuning procedure but, heuristically, for the algorithm to have an anticipatory character, $f < \Delta_u \leqslant \Delta_1$ and $f < \Delta_u \leqslant \Delta_2$ should hold.

Creation of bundles and delivery routes. Once a system-wide target bundle size Z_t has been determined, the set $S = \bigcup_{r \in R} S_r$ of bundles to be assigned is built by processing the upcoming orders at each restaurant r, $U_{t,r}$, following the steps described in Procedure 3.

Procedure 3: Bundle generation using parallel-insertion

Input: $U_{t,r}$, set of upcoming orders at restaurant r,
Z_t, target bundle size,
and k_t^r, number of couriers that available at r (see Section 4.3.3).
Output: S_r, set of bundles from restaurant r to be assigned to couriers.

```
/* Initial construction                                          */
```

Sort the orders in $U_{t,r}$ by non-decreasing ready time
Compute the target number of bundles to create, $m_r = \max\left(k_t^r, \lceil |U_{t,r}|/Z_t \rceil\right)$
Initialize S_r with empty routes (bundles) $s_1, s_2, \ldots, s_{m_r}$
for $o \in U_{t,r}$ **do**
    repeat
        Find the route $s \in S_r$ and the insertion position i_s for order o into route s which results in the minimum increase in route cost, where the route cost of s is $\sum_{(p,q)\in s} TravelTime(p,q) + \beta \sum_{p\in s} ServiceDelay(p)$
    until $|s| < Z_t$ or insertion improves route efficiency
    Insert o in route s at position i_s

```
/* Improvement by ''remove-reinsert'' local-search               */
```

for $o \in U_{t,r}$ **do**
    Remove o from its current route;
    Given the existing routes in S_r, find route s and position i_s to re-insert o at minimum cost;
    Re-insert o into route s at position i_s

Note that once a bundle reaches its target size, an additional order is inserted only if this increases route efficiency, i.e., the time per order delivered in that bundle decreases. Also note that the parameter β, which controls the importance of freshness in the construction of bundles, is given beforehand (*i.e.*, it should be tuned off-line).

4.3.2 Assignment logic

At the core of our algorithm, the solution of a bipartite matching problem assigns order bundles to couriers, dictating the next delivery route to be executed by each courier. Such a simple model is able to capture and balance the trade-off between short-run efficiency and service quality, while ensuring that, in practical implementations, optimization is not a significant bottleneck in the overall solution process. But, of course, simplicity comes

at the cost of expressiveness: the only levers to guide decisions in the matching are the weights and feasibility conditions of order-courier pairs, and only so many aspects of the problem can be captured by these. In an attempt to retain some more control in the assignment process, specifically around the issue of how to avoid service delays for bundles that are already late, we introduce a priority scheme: assignments are built by first partitioning the set of relevant bundles in priority groups based on their "urgency", and then finding optimal assignments sequentially for each group.

Priority scheme. Orders are classified according to their unavoidable delays in drop-off and pick-up:

- **Group I**: orders whose target drop-off time is impossible to achieve.
- **Group II**: orders not in I which cannot be feasibly picked up at their ready time.
- **Group III**: orders that do not fall into the prior categories.

A bundle is assigned the highest priority of any of its constituent individual orders.

By creating assignments sequentially for these priority groups, urgent deliveries are more likely to achieve an earlier pickup time than if all orders were included in one big matching problem, which may hopefully make a difference at the time of commitment, preventing the postponement of delivery for orders that are already late. Similarly, in matchings where there are less couriers available than bundles to be assigned, a solution obtained with this priority scheme averts situations where orders that are already late do not get a courier assigned.

Three alternative optimization models, with increasing degrees of complexity, are considered.

Linear assignment model

The simplest assignment model, and the one used by default in our solution algorithm, is a bipartite matching with no side constraints, *i.e.*, a linear program. To define it, we introduce the following notation:

- N_s: number of individual orders in bundle s,
- θ: constant "penalty" for delays in the pick-up of an order or bundle of orders,
- $\pi_{s,d}$: pick-up time of bundle s if assigned to courier d. Note that, by definition, $\pi_{s,d} \geq \max_{o \in s}\{e_o\}$.
- $\delta^s_{o,d}$: order drop-off time of order o in bundle s if s is assigned to courier d (note that this value depends on the time that d can pick up s).
- $x_{s,d}$: 0-1 decision variable for the assignment of bundle s to courier d.

Using these definitions, the problem is:

$$\max_{x} \sum_{d \in D_t} \sum_{s \in S} \left(\frac{N_s}{\max_{o \in s}\{\delta^s_{o,d}\} - e_d} - \theta \left(\pi_{s,d} - \max_{o \in s}\{e_o\} \right) \right) x_{s,d} \tag{4.1a}$$

$$\text{s.t.} \sum_{s \in S} x_{s,d} \leqslant 1, \ \forall\, d \in D \tag{4.1b}$$

$$\sum_{d \in D \cup \{0\}} x_{s,d} = 1, \ \forall\, s \in S \tag{4.1c}$$

$$x_{s,d} \in \{0,1\}, \ \forall\, s \in S, d \in D \cup \{0\} \tag{4.1d}$$

Observe that the first term in the matching weights captures a "throughput" value, dividing the number of orders in a delivery route, N_s, by the time needed to complete the assignment, $\max_{o \in s}\{\delta^s_{o,d}\} - e_d$. Meanwhile, the second term relates to the assignment pick-up delay that causes a loss of freshness in meals, $\pi_{s,d} - \max_{o \in s}\{e_o\}$. The model formulation makes use of an artificial courier collecting excess bundles (by defining these artificial assignments to have a value strictly lower than the value of all actual feasible

assignments).

More complex integer programming assignment models

A relatively simple but powerful departure from the previous formulation is to drop the requirement that the set of bundles in the assignment problem be pairwise disjoint with respect to the set U of orders to be assigned. Instead, that condition can be enforced as a constraint in the optimization problem. The freedom thus gained can be leveraged to mitigate some potential pitfalls inherent in the process of assigning (at most) one bundle to each courier on every optimization run.

Let $\mathbb{S}$ be the set of bundles considered for assignment, and define $\mathcal{Q} \subseteq \mathbb{S} \cup \mathbb{S} \times \mathbb{S}$ as the set of one or two bundles that can be assigned to a courier (when assigned two bundles, the courier completes one after the other in the prescribed sequence). For any order o, denote the set of routes that contain o by $\mathcal{Q}(o)$. Finally, define variables:

$$x_{q,d} \in \{0,1\} : \text{indicates whether route } q \text{ is assigned to courier } d, \text{ for } q \in \mathcal{Q},\ d \in D \cup \{0\}$$

$$y_o \in \{0,1\} : \text{indicates whether order } o \text{ is assigned to any courier, for } o \in U$$

Using these definitions, the assignment problem can be formulated as:

$$\max_x \sum_{d \in D} \sum_{q \in \mathcal{Q}} w_{q,d} x_{q,d} + \sum_{o \in O} p(1 - y_o) \tag{4.2a}$$

$$\text{s.t.} \sum_{q \in \mathcal{Q}} x_{q,d} \leqslant 1 \qquad \forall\ d \in D \tag{4.2b}$$

$$\sum_{d \in D \cup \{0\}} x_{q,d} = 1 \qquad \forall\ q \in \mathcal{Q} \tag{4.2c}$$

$$\sum_{d \in D} \sum_{q \in \mathcal{Q}(o)} x_{q,d} \leqslant 1 \qquad \forall\ o \in U \tag{4.2d}$$

$$\sum_{d \in D} \sum_{q \in \mathcal{Q}(o)} x_{q,d} \geqslant y_o \qquad \forall\ o \in U \tag{4.2e}$$

$$x_{q,d} \in \{0,1\} \qquad \forall\, q \in \mathcal{Q}, d \in D \cup \{0\} \tag{4.2f}$$

$$y_o \in \{0,1\} \qquad \forall\, o \in U \tag{4.2g}$$

Constraints (4.2b) guarantee that each courier is assigned at most one route, while (4.2c) guarantee that each route is assigned to one courier (it may be 0, a "dummy" courier). Constraints (4.2d) guarantee that each order is in exactly one of the routes assigned. If an order $o \in O$ is not included in any of the routes assigned to real couriers, a value p is added to the objective value (with $p < \min_{q,d}\{w_{q,d}\}$, to discourage null order assignments if at all possible, *i.e.* to assign as many orders as possible to real couriers). The weight of a (q,d) assignment is $w_{q,d} = u_{s,d}$ if q consists of only one bundle s, or $w_{q,d} = u_{s,d} + v_{s,s',d}$ if q consists of a sequence of bundles (s,s'), where u and v are defined as:

$$u_{s,d} = \frac{N_s}{\max_{o \in s}\{\delta^s_{o,d}\} - e_d} + \theta(\pi_{s,d} - \max_{o \in s}\{e_o\}) \qquad \forall s \in \mathcal{S},\, d \in D \tag{4.3}$$

$$v_{s,s',d} = \frac{N_{s'}}{\max_{o \in s'}\{\delta^{s,s'}_{o,d}\} - \max_{o \in s}\{\delta^s_{o,d}\}} + \theta(\pi_{s,s',d} - \max_{o \in s'}\{e_o\}) \qquad \forall (s,s') \in \mathcal{Q},\, d \in D \tag{4.4}$$

Here N_s and $N_{s'}$ denote the number of orders in bundles s and s', respectively; e_d denotes the time that courier d is available to begin a new route; $\max_{o \in s}\{\delta^s_{o,d}\}$ is the time of the last delivery associated to s if d performs the delivery; $\max_{o \in s'}\{\delta^{s,s'}_{o,d}\}$ is the time of the last delivery associated to s' if this bundle is served by d immediately after completion of deliveries of s; $\max_{o \in s}\{e_o\}$ is the time that s is ready for pick-up; $\pi_{s,d}$ is the earliest possible time that d can pick up s; and $\pi_{s,s',d}$ is the earliest possible time that d can pick up s' after having picked up and delivered s.

Defining $\mathcal{Q}$ by concatenating compatible "follow-on" bundles. The proposed construction procedure of set $\mathcal{Q}$ is designed to mitigate a potential pitfall of the myopic ap-

proach, arising when two bundles are assigned to different couriers even though they could have been delivered more efficiently by a single courier without degrading service quality. Given a set of basic bundles, $\mathbb{S}$ (where pairs of bundles are allowed to have orders in common), $\mathcal{Q}$ is constructed by finding all pairs of bundles in $\mathbb{S}$ that can be concatenated in such a way that second bundle does not suffer an excessive freshness loss. We define a freshness loss tolerance $\lambda \geqslant 0$, and follow Procedure 4.

Procedure 4: Find compatible "follow-on" bundle pairs

Input: Ground set of bundles $\mathbb{S}$, freshness loss tolerance $\lambda \geqslant 0$.
Output: Set $\mathcal{Q} \subseteq \mathbb{S} \cup \mathbb{S} \times \mathbb{S}$ of routes to be assigned to couriers.
Initialize $\mathcal{Q} \leftarrow \mathbb{S}$
for $s \in \mathbb{S}$ **do**
  for $s' \in \mathbb{S} \setminus \{s\}$ **do**
    if $s \cap s' = \emptyset$ **then**
      Compute $\pi_{s,s',d}$
      if $r_{s'} + \lambda \geqslant \pi_{s,s',d}$ **then**
        $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(s, s')\}$

Note that if $\lambda = 0$, only pairs of bundles where the second pick-up can be done at the earliest possible time are considered compatible. In contrast, if λ is large enough, any pair of bundles with no orders in common is a compatible "follow-on" pair.

Alternative definitions of $\mathbb{S}$. The ground set to produce $\mathcal{Q}$ can simply be the set S of bundles generated by Procedure 3, or it may be larger. A simple and effective way of creating more routes is to break each bundle of S into two pieces, as described in Procedure 5.

The motivation of Procedure 5 is to prevent the assignment of large bundles in S (which are less likely to be ready for immediate pick-up) from unnecessarily delaying the dispatch of orders that could be delivered soon (bundles in $\mathcal{S}^1$), while at the same time attempting to find a courier (and, if necessary, make a partial commitment) for orders that will not be ready as soon (bundles in $\mathcal{S}^2$).

Procedure 5: Split original bundles by their ready times

Input: Set of bundles S.
Output: $\mathbb{S}$, ground set of not necessarily order-disjoint bundles.
Initialize $\mathcal{S}^1 \leftarrow \emptyset$, $\mathcal{S}^2 \leftarrow \emptyset$
for $s \in S$ **do**
  Sort orders in s by increasing ready time, and relabel as $o_1, \ldots, o_k$ (o_k is ready the latest)
  for $j \in \{1, \ldots, k-1\}$ **do**
    Create bundle s_j^1, containing $o_1, \ldots, o_j$, the first j orders to be ready from s. Preserve sequence of deliveries originally prescribed in s for these orders
    $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \cup s_j^1$
    Create bundle s_j^2, containing $o_{j+1}, \ldots, o_k$, which are ready after o_j. Preserve the sequence of deliveries originally prescribed in s for these orders
    $\mathcal{S}^2 \leftarrow \mathcal{S}^2 \cup s_j^2$
$\mathbb{S} \leftarrow S \cup \mathcal{S}^1 \cup \mathcal{S}^2$

Assignment model variations. Two alternative models are explored in addition to the linear model:

(i) A "medium complexity" model, consisting of integer program 4.2 and $\mathcal{Q}$ defined by Procedure 4 acting on S, the bundles created by Procedure 3.

(ii) A "high complexity" model, consisting of integer program 4.2 and $\mathcal{Q}$ defined by Procedure 4 acting on $\mathbb{S}$, the set of bundles resulting from Procedure 5.

4.3.3 Commitment strategy

One of the ways that a rolling horizon algorithm attempts to mitigate uncertainty is through the postponement of decisions that are not time-critical. In light of the characteristics of the MDRP, our we adopt a "lazy" strategy that decomposes the information from assignments into two travel segments: "inbound" travel to the restaurant, and an "outbound" delivery trip (*i.e.*, an open route).

Two-stage additive commitment. For each tentative assignment (s, d), of order bundle s and courier d, in the solution of a matching problem, the commitment strategy dictates:

1. If d can reach restaurant r_s before $t + f$ and all orders in s are estimated to be ready by $t + f$, make a *final commitment* of d to s: instruct d to travel to r_s, pick up and deliver orders in s.
2. If d cannot reach r_s by $t + f$, but completes its last assignment before $t + f$, make a *partial commitment*: instruct d to travel to r_s and wait there for a finalized order assignment, which is guaranteed to include orders in s, and possibly more.
3. If d cannot start a new assignment by $t + f$, ignore the assignment.
4. *Exception:* If any order in s has been ready for more than x minutes, override the rule and make a final commitment.

The motivation to send a courier to the restaurant without committing the delivery of a specific bundle – as would be the case in a simpler single-stage commitment rule – is that, even if travel should begin without delay, the courier can be matched again in the next optimization, while en-route, and the composition of the bundle to be picked up at the restaurant may change. On the other hand, if the courier is busy before $t + f$, waiting for the next optimization will not delay the pick-up or delivery of any order.

We call this strategy "additive" because if s is partially committed to d at time t, then at optimization time $t + f$, we force the bundle assigned to d to include s. Hence, orders in a bundle partially assigned to d are guaranteed to be in the bundle finally assigned to d. Of course, this is not the most flexible policy that can be conceived, as one could completely decouple inbound and outbound assignments, but we decide to adopt it because in practical applications there are good arguments for such consistency (*e.g.*, to mitigate the risk of couriers rejecting time-critical assignments).

Two-stage additive commitment logic for assignments with more than one bundle. In the medium and high complexity assignment models, when dealing with the assignment of courier d to route $q = (s_1, s_2)$, the commitment rule is generalized in a way that delays final decisions on the actions to be taken after the completion of s_1, so long as this does not generate a decrease in the service quality of orders in s_2. Concretely, given a tentative assignment of order batch $q = (s_1, s_2)$ to courier d, our commitment strategy dictates:

1. If d can complete the delivery of s_1 *and* begin the delivery of s_2 by time $t + f$, make a final commitment to both s_1 and s_2.
2. If d can complete the delivery of s_1 by $t + f$ but cannot begin the delivery of s_2 by $t + f$, make a final commitment to s_1 and a partial commitment to s_2.
3. If d can only begin the delivery of s_1 by $t + f$, *i.e.*, if d can reach restaurant r_{s_1} by $t + f$ and all orders in s are estimated to be ready by $t + f$, make a *final commitment* of d to s_1: instruct d to travel to r_{s_1}, pick up and deliver orders in s_1.
4. If d cannot pick up s_1 by $t + f$, but completes its last assignment before $t + f$, make a *partial commitment* for d: instruct d to travel to r_{s_1} and wait there for a finalized order assignment (guaranteed to include orders in s_1.
5. If d cannot start a new assignment by $t + f$, ignore the assignment completely.
6. *Exception 1:* If any order in s_1 has been ready for more than x minutes, make a final commitment to s_1.
7. *Exception 2:* If any order in s_2 has been ready for more than x minutes, make a final commitment to s_1 and s_2.

It is worth noting that our two-stage additive commitment strategy is consistent with the MDRP operating environment that allows assignment updates: a courier may be instructed to move to a restaurant as soon as there is at least one order for the courier to pick up, but before a final determination is made about the exact set or sequence of orders to be assigned. Furthermore, a single pre-positioning move is used by our algorithm: at the

beginning of each courier duty time, a randomized instruction is handed to the courier, telling him to move to a nearby restaurant (the rest of instructions are deterministic, only making use of information already revealed).

4.4 A courier shift scheduling algorithm

4.4.1 The shift cover problem.

Before we introduce our offline shift-optimization process, we begin by describing the optimization problem that lies at its core.

Let the planning horizon be $[0, T]$. An activity profile is a right-continuous function $a : [0, T] \to \mathbb{Z}_{\geq 0}$ indicating at each point in time t an activity level $a(t)$ that needs to be covered by resources (e.g., at each point in time it indicates the number of active drivers required). Furthermore, let there be a finite number of points $0 < t_1 < t_2 < \ldots < t_n < T$ at which the activity level changes, either up or down. We assume that $a(t) = 0$ for $0 \leq t < t_1$ and for $t_n \leq t \leq T$.

Resources can be allocated to perform activities in the form of shifts. A resource profile is a function $r : [0, T] \to \mathbb{Z}_{\geq 0}$ indicating at each point in time the number of active resources, where the number of active resources is determined by the number of active resource shifts. A resource profile is feasible if $a(t) \leq r(t)$ for $t \in [0, T]$. Each resource shift has to satisfy the following requirements:

1. The length of any shift must be a number from $L = \{l_1, \ldots, l_p\}$, where $l_1 \leqslant \ldots \leqslant l_p$.
2. A shift can start only at pre-specified shift start times $0 < s_1 < \ldots < s_m < T$, with $s_1 \leq t_1$ and $s_m + l_1 \geq t_n$.

The shift cover problem is to minimize the total shift length required to cover the activity profile, i.e., to minimize the area under a feasible resource profile, given the start times and the shift lengths allowed.

If we denote the number of shifts of length l_k that start at time s_j by $x_k^{s_j}$, then the following integer program solves the shift cover problem.

$$\min \sum_{j=1}^{m} \sum_{k=1}^{p} l_k x_k^{s_j}$$

$$s.t. \sum_{k=1}^{p} \sum_{t_i - l_k \leqslant s_j < t_i} x_k^{s_j} \geqslant a(t_{i-1}) \qquad \forall i = 1, \ldots, n$$

4.4.2 A concrete implementation

To decide the courier schedule to implement on a given instance, we can compute an "activity profile" based on the solution to a perfect-information pick-up and delivery problem with time windows, obtained via the Adaptive Large Neighborhood Search (ALNS) heuristic [65]. The activity profile counts the number of non-idle drivers in the perfect-information solution at regular intervals throughout the operating period(*e.g.*, every optimization time). Then, we can follow Algorithm 6.

4.5 Computational study

In this section, we describe the design and results of a computational study to assess the quality of solutions produced by our algorithm (in its different variations), and how this performance is related to instance characteristics and key algorithmic features. A larger set of visualizations of our instancesis available in the Appendices.

4.5.1 MDRP instances

Instances have been crafted to resemble realistic day-long order and courier patterns in metropolitan areas, based on historical data provided by our industry partner for different cities and days. We create a total of 240 instances of varying sizes, ranging from 242

Algorithm 6: Shift optimization procedure based on perfect-information heuristic solution

Data: $\mathbb{P}$, activity profile computed from ALNS solution to perfect-information PDPTW.
B, the maximum total courier hours used historically.
Result: Optimized shift schedule for the dynamic problem

$a \leftarrow 1$ `// value by which to re-scale the ALNS activity profile`
repeat
  $\mathbb{P}' \leftarrow$ scale up $\mathbb{P}$ by a `// multiply value counts of every period in` $\mathbb{P}$ `by` a
  Solve shift cover problem on $\mathbb{P}'$
  if *shift-length of* $\mathbb{P}'$ *exceeds* B **then** STOP repetition loop
  $a \leftarrow a + 0.1$
  $\mathbb{P} \leftarrow \mathbb{P}'$
until $B <$ *total courier hours in schedule induced by* $\mathbb{P}$
Locate courier start locations randomly around the busiest restaurants in the next hour from the starting time of each shift
return shift schedule induced by (re-scaled) $\mathbb{P}$, with total courier hours no larger than B

to 3213 orders, 54 to 323 restaurants, 53 to 457 couriers, and 123 to 1542 courier hours. Times are represented as non-negative integers (with zero representing the start of business hours). Locations have been anonymized while preserving their overall distribution, and are represented as (x, y) coordinates in meters from a reference point. The travel time from $\ell_1 = (x_1, y_1)$ to $\ell_2 = (x_2, y_2)$ is the product of their Euclidean distance and a multiplier γ, rounded up to the next integer: $t_{\ell_1,\ell_2} = \left\lceil \gamma \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \right\rceil$. Note that this may lead to cases where the triangle inequality for travel times does not hold.

Service times at all restaurants are set to 4 minutes, and service times at all order delivery locations are likewise set to 4 minutes. The click-to-door target is 40 minutes in all instances, and the maximum allowed click-to-door is 90 minutes. Couriers are paid $10 per order delivered and are promised a $ 15 minimum compensation per hour.

For each "seed" instance, we have prepared a set of 24 variations:

- **Varying size of order and courier sets.** Given an order set, two instance variations are obtained by sampling 50% of the orders directly (uniform sampling with repetitions) or indirectly, through sampling of restaurants (where including a restaurant means including all the orders placed at it). Courier sets are reduced in a similar fashion, sampling roughly the same proportion of shifts at each start time (rounding up to the nearest integer number of shifts when necessary). These variations preserve some of the geographical and temporal distribution of orders. A more detailed exposition about the reduction procedures for order and courier sets is available in Appendix A.1.
- **Varying travel times.** Different location densities can be induced by using different γ multipliers. This preserves the relative spatial distribution of orders but, of course, impacts the nature of the solutions, as travel time changes (*e.g.*, orders served per courier hour, performance metrics associated with bundled-orders, etc.). Concretely, to build these instance variations, the original multiplier of each instance is reduced by 25%, producing a short travel time version.
- **Varying structure of courier schedules.** Schedules can resemble the patterns observed historically, or follow an "optimized" distribution. To ensure fairness in comparisons, optimized schedules preserve the total number of courier-hours of their historical counterparts. Optimized shift schedules are obtained following the procedure described in section 4.4.
- **Varying preparation times.** We can modify the degree of "visibility" of upcoming orders in an instance by changing the time it takes to prepare each meal, i.e. the difference between the time when an order is ready and its placement time, thus affecting the level of flexibility in the dispatching process. At the same time, given a target click-to-door time, longer preparation times imply that less time is available for a timely pickup once the order becomes ready. Furthermore, if instances feature

historical courier schedules, this transformation potentially creates a mismatch in the timing of courier capacity (in general, couriers turn out to be scheduled "too late" or "too early"). In our instance set, these instance variations are built by increasing the original preparation times by 25%.

To facilitate the study of different algorithms and solution strategies, our MDRP instances, together with a script to check feasibility and evaluate performance metrics of a solution, are available online at:

`https://github.com/ldamian21/MealDeliveryRoutingProblem.`

Details about instance and solution encoding can be found in Appendices B.2 and B.3.

4.5.2 Uncontrolled instance features

Beyond the features controlled during the instance generation process, we have focused our analysis on a set of structural properties whose importance has been long recognized in the dynamic routing literature [25]. We measure four features for each instance: geographic dispersion, dynamism, urgency and flexibility.

- **Geographic dispersion** captures the separation between restaurant and delivery locations over the instance geography. Other things equal, the more disperse locations are, the longer it takes to complete an average route, and the harder it becomes to build routes that achieve an acceptable performance (in order click-to-door, courier utilization, etc.). We measure geographic dispersion in terms of the travel time from restaurant to delivery location of any order, and the travel time between any pair of restaurants, using the instance mean and standard deviation across all relevant distance travel time measures. More precisely, we define the dispersion of an instance as:

$$dispersion = \frac{\sum_{a \in R, b \in R} t_{\ell_a, \ell_b}}{|R \times R|} + \frac{\sum_{o \in O} t_{\ell_{r_o}, \ell_o}}{|O|}$$

- **Dynamism** captures the continuity of change in the information available over the planning period. Under this definition, an instance where orders arrive in a few "bursts" is considered less dynamic than an instance where orders arrive at an even rate over time. For simplicity, we measure only the degree of dynamism of the arrival stream of orders (disregarding information from courier sign-on and sign-off events), using the method introduced in Lon, Ferrante, Turgut, Wenseleers, Berghe, and Holvoet [25], which we reproduce below for the sake of completeness. The degree of dynamism of an instance is always a number between 0 (no dynamism, *i.e.*, all orders are simultaneously revealed) and 1 ("maximal" dynamism), computed as follows.

  Let n be the number of orders placed and A be the non-decreasing sequence of order placement times. Let H be the sequence of inter-arrival times, $H = (\eta_1, \eta_2, \ldots, \eta_{n-1}) = (a_2 - a_1, \ldots, a_n - a_{n-1})$. Let $T = \min\{\max_{c \in C}\{l_c\}, a_n\} + \tau_{\max}$ be regarded as the operating period, and define $\varphi = \frac{T}{n}$ as the "perfect inter-arrival time" (corresponding to arrivals with maximum dynamism). Define the sequence of penalized inter-arrival deviations as

$$\sigma_0 = 0, \;\; \sigma_i = (\varphi - \eta_i + \frac{\varphi - \eta_i}{\varphi}\sigma_{i-1})^+ \text{ for } i = 1, \ldots, n-1$$

  Note that this measure penalizes "bursts" of arrivals over a short period. Then, the instance degree of dynamism is defined as:

$$dod = 1 - \frac{\sum_{i=1}^{n-1} \sigma_i}{\sum_{i=1}^{n-1} \bar{\sigma}_i}$$

where $\sum_{i=1}^{n-1} \bar{\sigma}_i$ is a normalizing constant defined by $\bar{\sigma}_i = \varphi + (\frac{\varphi - \eta_i}{\varphi}\sigma_i)^+$ (capturing the maximum possible penalized deviations).

- **Urgency** captures the range of time available to complete the delivery of orders in a satisfactory way. We define "soft" and "hard" measures, relative to target and maximum click-to-door values, respectively. For clarity, we use "reaction time" measures, negatively related to urgency: the higher the reaction time of orders, the less urgency in the instance. As in [25], we summarize the urgency level of an instance by the mean of individual order reaction times:

$$react_{soft} = \frac{\sum_{o \in O} \tau - t_{\ell_{r_o}, \ell_o}}{|O|}$$
$$react_{hard} = \frac{\sum_{o \in O} \tau_{\max} - t_{\ell_{r_o}, \ell_o}}{|O|}$$

- **Flexibility** is closely related to urgency, but not completely equivalent. It captures the effective range of time available to dispatch an order, if this is going to be delivered in a satisfactory way. Measuring flexibility is important in meal delivery, because relatively long preparation times (with respect to service targets and guarantees) can make it hard to deliver an order on time, even if there is a large reaction time: if the order is not ready, a part of this reaction time is useless. We measure flexibility as follows:

$$flex_{soft} = \frac{\sum_{o \in O} (a_o + \tau) - (e_o + t_{\ell_{r_o}, \ell_o})}{|O|}$$
$$flex_{hard} = \frac{\sum_{o \in O} (a_o + \tau_{\max}) - (e_o + t_{\ell_{r_o}, \ell_o})}{|O|}$$

4.5.3 Questions for analysis

Our computational study serves three main purposes: i) measure the performance effects of controlled changes in instance features, ii) measure the relationship between performance and uncontrolled instance features, and iii) evaluate the impact of key algorithmic design decisions. Concretely, our experiments provide evidence to evaluate the following hypotheses and open-ended questions.

Performance effect of directly controlled instance features:

- **Effect of instance size reductions.** Instances created through a 50% reduction tend to have more dispersion (over space and time), which may have an effect on the potential benefits of bundling. We can compare the performance of the algorithm on each of the three instance subsets (no reduction, 50% reduction applied directly on order set, 50% reduction applied indirectly through the restaurant set), and verify whether reducing the instances significantly affects performance. Furthermore, since instances where the reduction preserves all orders for each selected restaurant should have more consolidation opportunities than reduced instances where this is not the case, we expect to see this reflected in a performance advantage of the former over the later.
- **Effect of faster travel times.** Slower travel times reduce reaction time and flexibility, and increase the time it takes to complete any given route. We expect faster travel times to lead to better overall service performance and lower courier utilization levels.
- **Effect of longer preparation times.** Increasing preparation times reduces flexibility during the assignment process. Therefore, longer preparation times should lead to worse service performance.

Performance effects of uncontrolled instance features:

- What relationships can be observed between the performance of the algorithm and geographic dispersion, dynamism, urgency, flexibility, and instance size?
- Which individual structural property affects algorithmic performance the most?

Value of key algorithmic features:

- Which algorithmic features have the largest marginal impact in solution quality?
- How does the magnitude of the impact of each algorithmic feature change as a function of urgency, dynamism, flexibility, and dispersion? In particular,
    1. is there any advantage in using shorter optimization periods, in more urgent instances? in more dynamic instances?
    2. is there any advantage in finding assignments for orders that become ready farther in the future, in more urgent instances? in more dynamic instances?
    3. is the algorithm very sensitive to the choice of the look-ahead horizons for the bundle target size calculations? does the sensitivity increase in urgent instances? in more dynamic instances? in more geographically disperse instances?
    4. how much is lost in performance by using an algorithm that does not use bundling in instances with low geographic dispersion? with low urgency?
    5. how critical is to have a two-stage commitment strategy in instances with high urgency and low flexibility?
    6. how much is gained in solution quality by adopting more complex integer programming formulations for the assignment problem? in instances with high urgency and low dynamism?

4.5.4 Analysis variables

Apart from categorical identifiers for each set of instance variations, and in addition to dispersion and degree of dynamism, we select the most informative measures of urgency, flexibility, and size by exploring the correlations in the instance dataset.

It is not surprising to verify that the soft and hard versions of urgency and flexibility measures are correlated, as shown in Table 4.1. To simplify the analysis, we focus only on hard reaction time and soft pickup flexibility, the pair of urgency and flexibility measures with the smallest correlation.

Table 4.1: Correlation matrix of urgency and flexibility measures

measure	soft resp. time	hard resp. time	soft pickup flex.	hard pickup flex.
soft resp. time	1.00	0.62	0.89	0.94
hard resp. time	0.62	1.00	0.41	0.46
soft pickup flex.	0.89	0.41	1.00	0.96
hard pickup flex.	0.94	0.46	0.96	1.00

Similarly, number of orders and total courier hours are highly correlated, as shown in Table 4.2. Since courier hours in the instances were been decided in advance, based on forecasts about the number and distribution of orders, we decide to measure instance size by the number of orders, discard total courier hours from the analysis variables, and include the ratio of orders to courier hours, which indicates how "busy" or congested an instance turns out to be (instances with a unusually high ratio suggest that the forecast might have been an underestimation, or perhaps there were not enough couriers to keep up with expected demand).

Having defined the analysis variables, let us briefly summarize the structural properties of the instance set. The histograms in the diagonal of Figure 4.1 illustrate the marginal distribution of the uncontrolled instance characteristics (measures aggregated in 8 buck-

Table 4.2: Correlation matrix of orders, courier hours, orders per courier hour and dynamism

measure	orders	courier hours	orders/courier hour	dynamism
orders	1.00	0.90	0.31	-0.31
courier hours	0.90	1.00	-0.06	-0.19
orders per courier hour	0.31	-0.06	1.00	-0.20
dynamism	-0.31	-0.19	-0.20	1.00

ets), and the hexagonal bins in the lower triangle of Figure 4.1 illustrate their pairwise-joint distribution.

Instances have degrees of dynamism ranging between 0.3 and 0.6, more frequently in the 0.4-0.55 range, and the most significant interactions of dynamism are with size (largest instances are also the least dynamic) and geographic dispersion (less disperse instances tend to be less dynamic). Most instances in the set have less than 1000 orders and a relatively dense geography (in most instances, individual-order inbound and outbound trips would on average total less than 20 minutes). Hard reaction time values range from 80.7 to 84.5 minutes on average, and show some correlation only with geographic dispersion: this is not unexpected, as both measures are a function of direct travel times from restaurant to delivery location. Soft pickup flexibility, ranging from 10.4 to 19.4 minutes, exhibits no strong correlation to any other feature. The ratio of orders placed to courier hours, ranging from 1.2 to 2.5, shows little correlation with all other instance features, but it is important to note that the largest instances (2500 or more orders) all have a very high ratio.

4.5.5 Experiment runs

For each instance, we obtain 21 solutions by running slightly different versions of the rolling horizon algorithm: the default algorithm, 5 variations differing in one feature only,

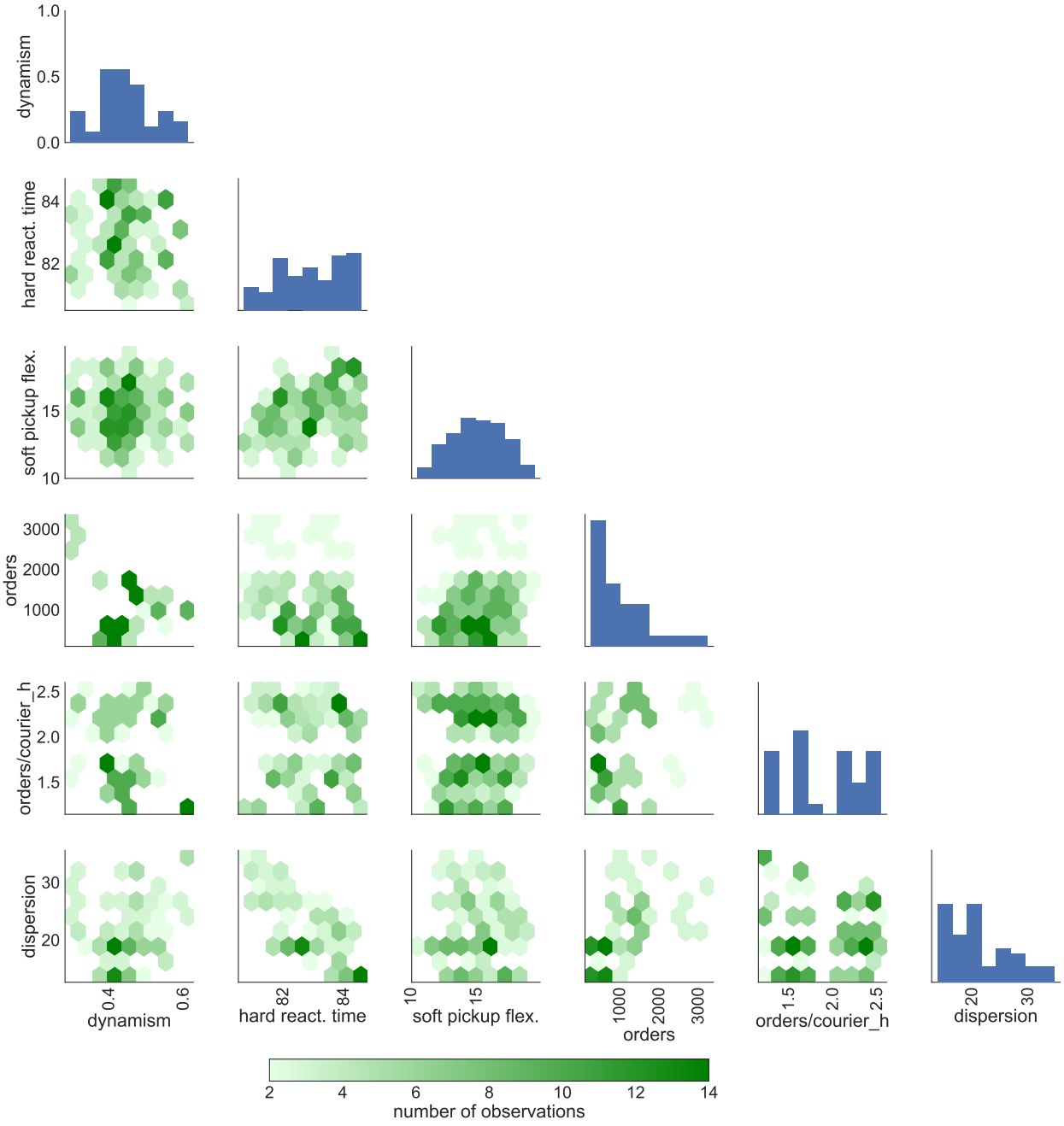


Figure 4.1: Distribution of key instance features

and 15 variations with alternative values of optimization frequency, assignment horizon, order and courier bundling intensity lookaheads. In detail, the algorithm variations explored are defined as follows:

Different optimization frequencies. Set the time between successive optimizations to $f = 5$ or $f = 2$ minutes (all other settings at their default values).

Different order-assignment horizon and myopic lookaheads. Experiment setting a lookahead of $2f$ or $4f$ (default) minutes for the numerator (orders) of the target size ratio calculation; setting a lookahead of $2f$ or $4f$ (default) minutes for the denominator (couriers) of the target size ratio calculation; and setting a horizon of $2f$ (default) or $4f$ minutes to limit the set of orders open for assignment. In sum, for each value of f, 8 configurations are considered.

Bundling intensity rules. Explore two algorithm variations, one using dynamic target bundle sizes (default), and the other explicitly forbidding any consolidation (*i.e.*, always assign couriers to single orders).

Assignment prioritization. Enforce three order priority groups in the assignment process (default), or find assignments for all orders in a single matching problem per optimization run.

Commitment. Follow the two-stage additive commitment strategy (default), or use a single-stage lazy strategy, where no more changes can be made in the "outbound" portion of the assignment as soon as couriers begin the"inbound" movement to the restaurant.

Complexity of assignment model. Consider the three different models described in Section 4.3.2 to govern the assignment process.

Table 4.3: Algorithm variations

Variation id	Description
0	Default ($f = 5, \Delta_u = 10, \Delta_1 = 10, \Delta_2 = 10$, two-stage commitment, 3 priority groups, dynamic bundling, LP assignment model)
1	$f = 5, \Delta_u = 10, \Delta_1 = 10, \Delta_2 = 20$
2	$f = 5, \Delta_u = 10, \Delta_1 = 20, \Delta_2 = 10$
3	$f = 5, \Delta_u = 10, \Delta_1 = 20, \Delta_2 = 20$
4	$f = 5, \Delta_u = 20, \Delta_1 = 10, \Delta_2 = 10$
5	$f = 5, \Delta_u = 20, \Delta_1 = 10, \Delta_2 = 20$
6	$f = 5, \Delta_u = 20, \Delta_1 = 20, \Delta_2 = 10$
7	$f = 5, \Delta_u = 20, \Delta_1 = 20, \Delta_2 = 20$
8	$f = 2, \Delta_u = 4, \Delta_1 = 4, \Delta_2 = 4$
9	$f = 2, \Delta_u = 4, \Delta_1 = 4, \Delta_2 = 8$
10	$f = 2, \Delta_u = 4, \Delta_1 = 8, \Delta_2 = 4$
11	$f = 2, \Delta_u = 4, \Delta_1 = 8, \Delta_2 = 8$
12	$f = 2, \Delta_u = 8, \Delta_1 = 4, \Delta_2 = 4$
13	$f = 2, \Delta_u = 8, \Delta_1 = 4, \Delta_2 = 8$
14	$f = 2, \Delta_u = 8, \Delta_1 = 8, \Delta_2 = 4$
15	$f = 2, \Delta_u = 8, \Delta_1 = 8, \Delta_2 = 8$
16	single-stage commitment
17	no priority groups
18	no bundling
19	medium complexity assignment model
20	high complexity assignment model

Before conducting the full-scale experiments, reasonable default values for a series of secondary parameters in the algorithm must be found. We tune the algorithm with a hierarchical service objective: first ensure the delivery of a very high proportion of the orders placed, and then minimize average click-to-door. Details about the tuning process can be found in Appendix B.4.

4.5.6 Experiment results

As a preamble to our analysis of experimental results through the prism of the questions formulated in Section 4.5.3, we benchmark the solutions obtained by the algorithm on a sample of instances with relatively few orders and couriers with provably optimal perfect-information solutions reported in [66].

Comparison of algorithm with exact method

The exact solutions provided by [66] have been obtained by solving an integer program with the objective to minimize total click-to-door. They are all provably or practically optimal (maximum optimality gap is 0.16%, *i.e.*, a couple of seconds away from the lower bound). In the integer program, delivery of all orders is enforced as a feasibility condition. It must be noted that the IP formulation restricts bundle sizes to be at most 2, but given the small size of the instances, and the distribution of orders over time, we have reason to believe that the effect of this restriction is very small in magnitude.

As shown in Table 4.4, on this set of small instances, our algorithm produces high-quality solutions with respect to mean click-to-door, on average only 4% above the optimal value (about 70 seconds, in absolute terms). At the same time, mean ready-to-pickup performance of our algorithm is on average 112% (45 seconds) above the value of the exact solutions, even though this measure has not been directly optimized in [66]. These results illustrate the challenge of multi-objective optimization: while there is large room

Table 4.4: Comparison of solutions obtained by our algorithm and exact solutions

instance	orders	restaurants	couriers	courier hours	Rolling Horizon Heuristic CtD	Rolling Horizon Heuristic RtP	Exact Solution (minimizing total CTD) CtD	Exact Solution (minimizing total CTD) RtP	Heur/Exact CtD	Heur/Exact RtP
0o50t100s1p100	252	93	61	151	31.19	2.52	29.81	1.46	1.05	1.73
0o50t100s1p125					34.67	2.27	33.40	1.23	1.04	1.85
0o50t100s2p100			72	146	29.79	1.22	28.96	0.64	1.03	1.91
0o50t100s2p125					34.18	1.85	32.60	0.53	1.05	3.49
0o50t75s1p100			61	151	28.4	1.65	27.49	0.96	1.03	1.72
0o50t75s1p125					31.62	1.19	31.10	0.78	1.02	1.53
0o50t75s2p100			72	146	27.29	0.58	26.83	0.30	1.02	1.93
0o50t75s2p125					31.19	0.7	30.52	0.24	1.02	2.92
0r50t100s1p100	242	54	61	151	32.46	2.14	30.76	1.13	1.06	1.89
0r50t100s1p125					36.75	2.16	34.66	0.97	1.06	2.23
0r50t100s2p100			73	146	31.21	1.11	29.93	0.43	1.04	2.58
0r50t100s2p125					35.6	1.22	33.86	0.26	1.05	4.69
0r50t75s1p100			61	151	29.57	1.04	28.47	0.67	1.04	1.55
0r50t75s1p125					33.71	1.19	32.45	0.55	1.04	2.16
0r50t75s2p100			73	146	29.03	0.64	27.95	0.21	1.04	3.05
0r50t75s2p125					33.41	0.84	32.00	0.16	1.04	5.25
average					31.879	1.40	30.674	0.66	1.04	2.53

for improvement in ready-to-pickup performance, one must keep in mind that focusing on minimizing this measure does not guarantee further improvements in click-to-door.

More challenging large-scale instances may exhibit important differences from the small instances in this benchmark (*e.g.*, more bundling opportunities), and there is no guarantee that performance will follow a similar trend. That being said, these preliminary results are certainly encouraging.

Performance effects of controlled instance characteristics

Tables 4.5-4.8 summarize the impact of each of the controlled instance variations on performance metrics over all experiment runs (*i.e.*, over all algorithm variations). Figures B.1-B.14 in Appendix B.5 illustrate these controlled effects in more detail.

Table 4.5: Differences in performance of instances with optimized courier schedules vs. historical schedules

	courier shift schedules				paired differences	
	historical		optimized		(opt. - hist.)	
	mean	std	mean	std	mean	std
% undelivered	0.42	0.91	0.16	0.28	-0.25	0.90
CtoD mean	33.7	3.78	32.38	3.03	-1.32	1.46
CtoD 90%	50.81	6.35	48.9	5.24	-1.91	3.12
CtoD overage	3.21	1.84	2.64	1.36	-0.57	0.93
RtoP mean	2.79	2.08	1.64	1.09	-1.15	1.23
RtoP 90%	8.35	5.47	5.06	3.13	-3.28	3.31
utilization mean	0.63	0.12	0.64	0.12	0.01	0.02
utilization 10%	0.29	0.14	0.35	0.15	0.06	0.06
cost per order	11.58	1.34	11.22	1.15	-0.35	0.31
orders per bundle	1.11	0.07	1.08	0.06	-0.02	0.03

Table 4.5 shows that leveraging accurate predictive information with off-line optimization techniques, in order to schedule the right capacity at the right time, has the potential to yield significant improvements in all service metrics, while reducing performance variability, and even reducing costs. We note that historical shifts pack more orders per

bundle, which is consistent with the algorithm using bundles as a recourse to mitigate capacity shortages: the more couriers signed on at productive times of the day, the more effective capacity over time, and the smaller the target bundle sizes.

Table 4.6: Differences in performance of instances with travel times faster by 25% vs original

	travel time multiplier				paired diff.	
	100%		75%		(75% tt. -100% tt.)	
	mean	std	mean	std	mean	std
% undelivered	0.36	0.88	0.22	0.4	-0.14	0.66
CtoD mean	34.79	3.35	31.29	2.64	-3.51	1.33
CtoD 90%	52.21	5.93	47.51	4.84	-4.70	2.33
CtoD overage	3.54	1.8	2.31	1.18	-1.23	0.82
RtoP mean	2.95	2	1.48	1.04	-1.47	1.18
RtoP 90%	8.65	5.14	4.75	3.33	-3.90	2.60
utilization mean	0.68	0.12	0.6	0.11	-0.08	0.02
utilization 10%	0.35	0.16	0.28	0.13	-0.07	0.06
cost per order	11.34	1.25	11.47	1.27	0.13	0.17
orders per bundle	1.11	0.07	1.08	0.05	-0.02	0.03

Table 4.6 summarizes the performance sensitivity of the algorithm to vehicle speed. If travel times are 25% faster, orders undelivered can be reduced by more than half, while improving mean click-to-door times by 10%, and slashing down the overage mean by more than a third and the ready-to-pickup mean by almost half. On the other hand, however, total costs increase, as the work in the system increases (orders undelivered drop) but is distributed less evenly (utilization mean and 10th percentile drop) and the system is forced to spend more on minimum compensation complements. We believe this illustrates the downside of having "too many" drivers at the wrong times of the day: when travel times are slower, routes take longer to complete and, at any point in time, more vehicles need to be on the road (as orders have not slowed down their arrival and the same service standard must be met), thus leading the algorithm to pull couriers that are available at a remote location into the action. On the other hand, when travel times

are faster, there is a relative surplus of couriers but the algorithm has little incentive to put them to work if they are not in a central location, as this may degrade service quality and potential cost-savings related to the compensation scheme are not considered by the algorithm.

Table 4.7: Differences in performance of instances with preparation times slower by 25% vs original

	preparation time multiplier				paired differences	
	100% prep. time		125% prep. time		(125% pt. - 100% pt.)	
	mean	std	mean	std	mean	std
% undelivered	0.2	0.6	0.38	0.76	0.18	0.31
CtoD mean	31.1	2.86	34.98	2.94	3.88	0.47
CtoD 90%	46.53	4.9	53.19	4.84	6.66	1.20
CtoD overage	2.07	1.25	3.78	1.54	1.71	0.46
RtoP mean	2.21	1.74	2.22	1.77	0.01	0.31
RtoP 90%	6.69	4.72	6.71	4.78	0.02	1.09
utilization mean	0.64	0.12	0.64	0.12	0.00	0.01
utilization 10%	0.32	0.15	0.32	0.15	0.00	0.04
cost per order	11.41	1.25	11.39	1.27	-0.01	0.12
orders per bundle	1.09	0.06	1.1	0.06	0.00	0.01

Table 4.7 illustrates the sensitivity of performance with respect to preparation time. If orders take 25% longer to be ready, orders undelivered almost double, while click-to-door falls by 12% on average, driving up the overage mean up by 80%. Interestingly, longer preparation times do not deteriorate ready-to-pickup performance significantly. Despite completing less deliveries (thus reducing total delivery payments), cost per order improves only slightly, as a consequence of having couriers scheduled 'out-of-phase', logging-on for duty during periods of low productivity at the beginning of lunch and dinner times (thus increasing the total minimum compensation payments).

Table 4.8 illustrates the performance effect of halving the number of orders and courier hours, either by thinning the overall order arrival process (`o50`) or by sampling a large enough subset of restaurants (`r50`). By and large, instances of original size tend to have

Table 4.8: Differences in performance of instance size reductions vs original

	instance size reduction						paired differences			
	o100		o50		r50		(o50-o100)		(r50 - o100)	
	mean	std	mean	std	mean	std	mean	std	mean	std
% undelivered	0.2	0.4	0.38	0.95	0.28	0.57	0.19	0.80	0.08	0.39
CtoD mean	32.55	3.17	33.29	3.75	33.27	3.47	0.74	1.32	0.72	1.52
CtoD 90%	49.31	5.54	49.96	6.4	50.3	5.68	0.65	2.97	0.99	2.88
CtoD overage	2.75	1.47	2.98	1.81	3.05	1.62	0.24	0.81	0.30	0.76
RtoP mean	1.68	1.28	2.61	2.13	2.36	1.61	0.94	1.16	0.69	0.66
RtoP 90%	5.2	3.56	7.69	5.85	7.22	4.17	2.49	3.23	2.02	1.70
utilization mean	0.62	0.12	0.66	0.12	0.64	0.12	0.04	0.02	0.03	0.02
utilization 10%	0.28	0.14	0.35	0.15	0.32	0.15	0.07	0.06	0.04	0.06
cost per order	11.5	1.31	11.28	1.2	11.42	1.25	-0.22	0.19	-0.09	0.21
orders per bundle	1.1	0.06	1.06	0.05	1.12	0.07	-0.04	0.03	0.01	0.02

better service performance than their reduced counterparts (thanks to the higher location density), at a slightly higher cost (a product of actually executing more deliveries). Furthermore, we highlight that in reduced instances, at any point in time there tend to be less restaurants with orders in preparation, and it takes longer on average for couriers to drive to the next pick-up (even if delivery locations are at a similar average distance from their associated restaurant). This is consistent with the results that show that 90th percentile ready-to-pickup measures degrade considerably more than 90th percentile click-to-door measures when instances are reduced: the algorithm struggles to find couriers that can make it on time to pick-up, but can still build routes that mitigate the delay impact on click-to-door.

Focusing now on the difference between both reduction types, `r50` instances fare better than `o50` instances on orders undelivered, utilization and ready-to-pickup measures, while in the case of click-to-door, `r50` instances fare better on the average measure, but worse in worst-case measures. `o50` instances achieve a lower cost per order than `r50` instances, but only because the algorithm cannot complete as many deliveries. To interpret these trends, consider that `o50` instances tend to have less orders per restaurant than

r50 (and o100) instances, *i.e.*, less consolidation opportunities than r50 instances. Consistently, the results show that o50 instances make less use of bundling than the original instances (because it is difficult), while r50 instances use bundling more intensely than the original instances (because it is necessary, to cope with higher geographic dispersion) and o50 instances (because there are more opportunities).

Performance effect of instance structural properties

Figure 4.2 summarizes the marginal effect of each instance feature on the set of performance measures over all experiment runs with the default algorithm settings (We focus on this variant in order to simplify the analysis, as we have no direct control over the structural properties being studied). Figures B.15-B.18, available in appendix B.6, illustrate the second-degree interaction effects of key instance features with each performance measure explored in Figure 4.2.

As dynamism increases, utilization and orders per bundle tend to decrease, while cost per order tends to increase. This is expected, as consolidation opportunities that help defray delivery costs are more likely to be available when orders arrive in bursts. Except for a couple of outliers (illustrated in more depth in Table 4.9), the percentage of orders undelivered seems to be fairly insensitive to degree of dynamism. If anything, instances with very low and very high dynamism seem to achieve a small percentage of orders undelivered more consistently than instances with mid-range dynamism. A similar pattern is observable for click-to-door and ready-to-pickup measures, which suggests that that there is another factor at play, not discernible in Figure 4.1 and Figure 4.2, driving this behavior. On closer inspection, the outliers and the interaction plots in Appendix B.6 show that poor performance always corresponds to a high ratio of orders to courier hours. While, given the available data, we do not venture to advance a conclusive opinion on the cause for the understaffed plans, errors in forecasting or planning may be to

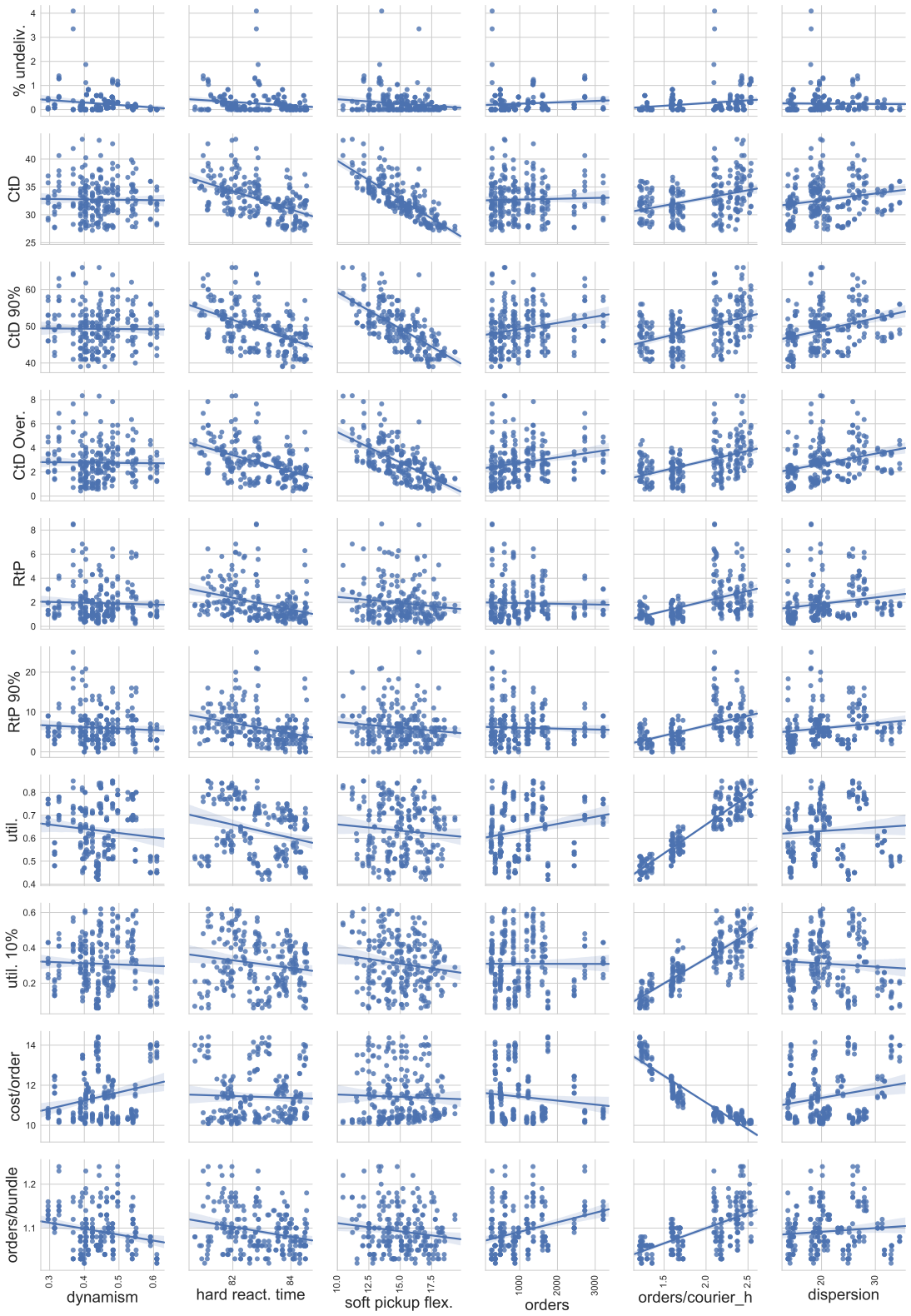


Figure 4.2: Effect of each instance feature on overall performance

blame.

As expected, the results show that urgency and flexibility have pronounced adverse effects in most measures, except cost per order. The linear trends in Figure 4.2 suggest that an additional minute of hard reaction time could reduce click-to-door roughly as much as 100 seconds in mean, 20 seconds in overage, and 2.5 minutes in the 90th percentile. Meanwhile, an additional minute of soft pickup flexibility could reduce click-to-door roughly as much as 1.25 minutes in mean, 30 seconds in overage, and 1.9 minutes in the 90th percentile. Given that urgency and flexibility are defined in terms of click-to-door target and maximum, choosing the right values for these parameters in realistic applications is of utmost importance.

The ratio of orders placed to courier hours scheduled affects all performance measures as expected: higher capacity relative to demand is tied to improvements in service performance. However, this improvement comes at a steep price: a decrease of the ratio from 2.5 to 1.25 orders per courier hour (*e.g.*, a doubling of the number of courier hours) increases costs by more than 3 cost units per order (*i.e.*, about 30%), pulling ready-to-pickup mean close to its lower limit of 0, but buying a reduction of less than 4 minutes in click-to-door mean (*i.e.*, around 8%), and 8 minutes in click-to-door 90th percentile (*i.e.*, around 15%).

Surprisingly, dispersion is not the strongest predictor of bundling intensity. In fact, out of the structural properties considered, dispersion has the weakest linear association to orders per bundle: an increase of 20 minutes in the dispersion measure brings the average number of orders per bundle up by 0.02 units only. Cost and service metrics exhibit a clearer linear association with dispersion, but still far from strong: an increase of 20 minutes in the dispersion measure translates, on average, to an increase of about 1 unit of cost per order, and an increase of 2.5 minutes in click-to-door mean, 6.5 minutes in click-to-door 90th percentile, and 1 minute in ready-to-pickup mean.

Interestingly, the size of instances does not affect ready-to-pickup mean and 90th per-

centile values, while the percentage of orders undelivered and click-to-door mean values suffer only a slight degradation in large instances. On the other hand, overage and 90th percentile click-to-door times do increase significantly with the number of orders placed. Moreover, the number of orders placed is positively correlated with the average size of bundles and average courier utilization. All this suggests that the algorithm fends off the challenge from larger instances by assigning larger routes, even if the click-to-door of the last orders delivered will suffer greatly, in order to achieve a better average performance.

However, it must be noted that in the case of the orders/courier hours ratio, its linear relationship with these performance metrics is much stronger, and (as shown in Figure 4.1) the relatively few instances with very large order sets have a very high ratio of orders to courier hours. Hence, size may not be the driver of bundling intensity and courier utilization (and service performance), but instead this is the balance between orders and courier hours.

From Figure 4.2 we have dug up the data to contextualize the abnormally high percentage of orders undelivered (beyond 3%) registered for two instances with a low degree of dynamism (in the 0.35-0.40 range). These instances have a very small size (269 orders), but a very high ratio of orders to courier hours, and their solutions achieved relatively high ready-to-pickup values, high utilization and a relatively low number of orders per bundle. Table 4.9 compares their performance vis a vis the corresponding instance variations featuring optimized schedules, and reveals that the historical courier capacity was not distributed in an effective way over time, and the system could not easily overcome larger than expected bursts of order arrivals. This example serves as a warning: no matter the size of the problems, the algorithm can only do so much in the presence of poorly distributed capacity, and, by the same measure, a good forecasting and courier scheduling process can go a long way towards guaranteeing reliable performance.

Most of the observable interaction trends in Figures B.15-B.18 are expected, given the

Table 4.9: Illustrating the potential consequences of inadequate courier schedules

instance	$\frac{\text{orders}}{\text{cour. h.}}$	% undel.	CtoD mean	CtoD 90%	CtoD overage	RtoP mean	RtoP 90%	util. mean	$\frac{\text{cost}}{\text{order}}$	$\frac{\text{orders}}{\text{bundle}}$
1o50t100s1p100	2.1	3.35	38.24	64	6.16	8.44	25	0.73	10.28	1.17
1o50t100s1p125	2.1	4.09	41.89	64.3	7.65	8.51	21	0.74	10.2	1.16
1o50t100s2p100	2.19	0	31.54	45.2	1.44	3.38	10	0.85	10.27	1.05
1o50t100s2p125	2.19	0	35.57	53	3.38	3.73	11	0.82	10.18	1.03

direction of the main effects. One result worth highlighting is the following: in scenarios with low pickup flexibility, low reaction time, or high dispersion, having higher dynamism enables a higher 10th percentile of courier utilization and a smaller percentage of orders undelivered. We conjecture that as orders are more evenly distributed over time in these difficult scenarios, the algorithm is able to find a few more assignments during non-peak times for otherwise poorly occupied couriers.

Performance value of algorithm features

The average performance of the different algorithm variations is summarized by a point estimate and a 95% bootstraped confidence interval (based on 1000 bootstrap iterations) in Figure 4.3, for variations 0-15, and Figure 4.3, for variations 16-21. Most variations, except 16 and 18, achieve an average percentage of orders undelivered similar to the default algorithm (0.25%). This suggests that allowing bundles and having a sensible commitment policy are the key to ensure that as many deliveries as possible are completed.

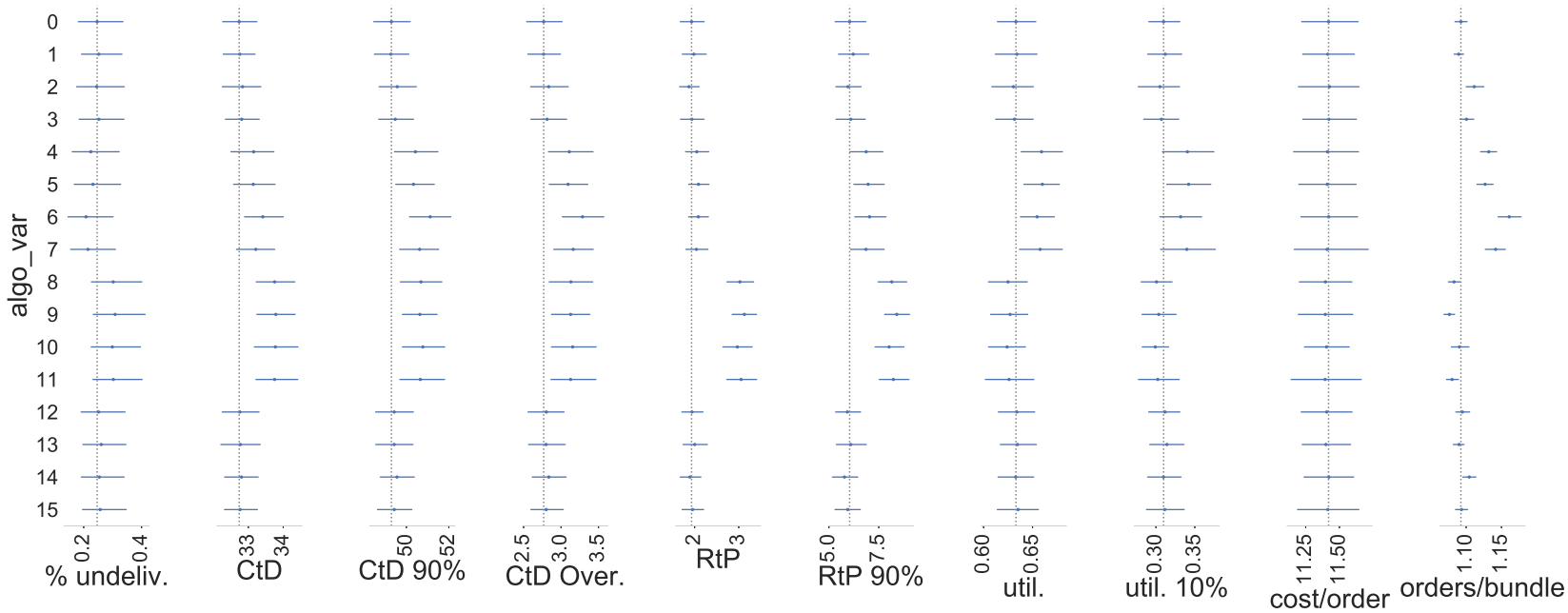


Figure 4.3: overall performance of each algorithm variation

Within variations 0-15, the average number of orders per bundle is quite sensitive to the specific configuration of parameters. Other things equal, bundling intensity tends to be higher for larger assignment horizons, and for the combinations that lead to higher target bundle sizes (long lookahead for orders and short lookahead for couriers), as expected. We highlight that once the rest of parameters have been decided, choosing myopic lookaheads for the target bundle size can have a significant but small effect on average performance.

Compared to the default algorithm, variations 4-7, which have a longer assignment horizon, achieve significantly worse service measures, and significantly higher utilization and bundling intensity: the assignment horizon is too long, leading to larger bundles (and higher utilization) that unnecessarily hurt the last deliveries in each route (worse click-to-door) and have a longer spread in ready times (worse ready-to-pickup).

Variations 8-11, with more frequent optimization runs, are also significantly outperformed by the default algorithm in all service metrics. In this case, however, there is not even a significant gain in utilization, which suggests that these variations are just too myopic. This conclusion is confirmed by the fact that variations 12-15, which still have more frequent optimizations compared to the default, but are relatively less myopic, achieve similar performance measures as the default algorithm.

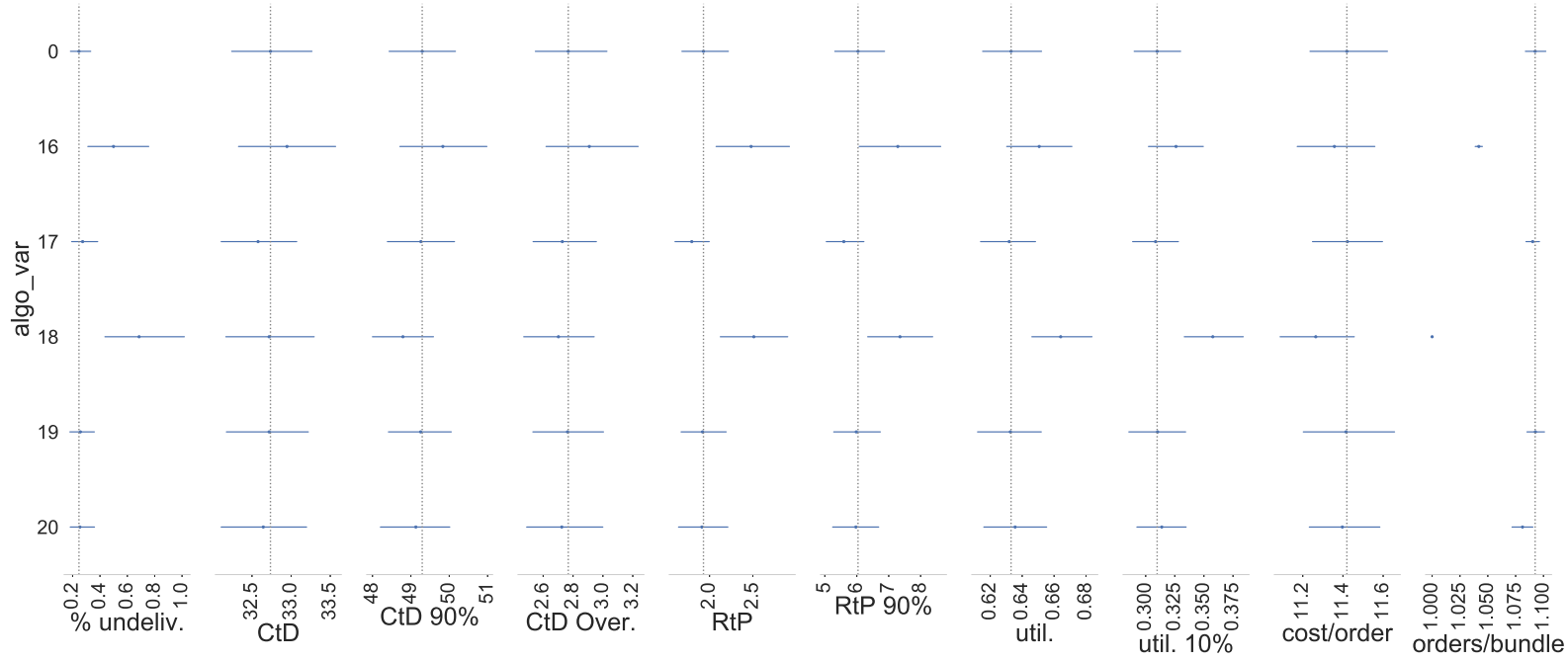


Figure 4.4: overall performance of main algorithm variations

As noted earlier, variations 16 (single-stage commitment strategy) and 18 (no bundling) lead to solutions with higher utilization values and less orders per bundle, due to less bundling opportunities. But this goes hand in hand with a significantly higher rate of orders undelivered, thus removing any cause for celebration in the apparent improvement in cost or the rest of service performance measures. What is worse, variation 16, with a single-stage commitment policy, also tends to increase click-to-door and ready-to-pickup measures.

Variation 17, where one matching, instead of three, is solved per optimization run, sacrifices a bit on the percentage of orders undelivered (0.27%) and cost per order (11.42), in order to outperform the default algorithm by a small margin in click-to-door mean (32.58), overage (2.73), and 90th percentile (49.26), as well as ready-to-pickup mean (1.79) and 90th percentile (5.59).

Variation 19, featuring the assignment model of medium complexity, is able to outperform the default algorithm, however slightly, on click-to-door metrics (32.72 mean, 2.76 overage, 49.25 90th percentile), ready-to-pickup (1.92 mean, 5.98 90th percentile), keeping cost per order in check (11.42) and paying only a small penalty in the percentage of orders undelivered (0.26%). This is achieved with orders per bundle and utilization values similar to the default algorithm.

As expected, variation 20, featuring the assignment model of highest complexity, is able to improve one more bit, achieving better click-to-door (32.64 mean, 2.72 overage, 49.13 90th percentile) ready-to-pickup (1.91 mean and 5.97 90th percentile) and cost per order (11.40), without increasing the percentage of orders undelivered (0.26%). This is achieved with slightly higher utilization (0.31 mean and 0.64 90th percentile), and slightly less orders per bundle (1.08).

A paired differences analysis (where we focus on the differences in performance from a particular variation and the default algorithm over all instances) can reveal more clearly

the value of each algorithmic feature, and allow us to assess how different instance structural properties amplify or reduce said value. Figures B.19-B.25, available in Appendix B.7, summarize the distribution of paired performance differences across the instance feature space.

Impact of optimization frequency. Table 4.10 summarizes the results of experiment runs in algorithm variations 0-16, paired by the configuration of horizon and lookahead multipliers, to compare two settings: one where there are 2 minutes between consecutive optimizations, and one where this interval is 5 minutes (default). The results show that, by and large, 5-minute optimization intervals lead to a lower percentage of orders undelivered, lower click-to-door mean, lower ready-to-pickup mean and 90th percentile, lower courier utilization mean and 10th percentile, and more orders per bundle than the alternative.

Table 4.10: Differences in performance of algorithm with more frequent optimizations (2 min) vs default (5 min)

	Optimization interval				paired differences	
	5 minutes		2 minutes		(f = 2) - (f = 5)	
	mean	std	mean	std	mean	std
% undelivered	0.23	0.48	0.28	0.51	0.05	0.18
CtoD mean	33.01	3.39	33.27	3.51	0.26	0.93
CtoD 90%	50.02	5.91	50.07	5.73	0.06	1.86
CtoD overage	2.98	1.63	2.97	1.63	-0.01	0.51
RtoP mean	2	1.49	2.5	1.79	0.50	0.78
RtoP 90%	6.5	4.58	7.07	4.35	0.57	2.07
utilization mean	0.65	0.12	0.63	0.12	-0.02	0.02
utilization 10%	0.32	0.16	0.31	0.13	-0.02	0.06
cost per order	11.42	1.3	11.4	1.21	-0.01	0.18
orders per bundle	1.12	0.07	1.09	0.05	-0.03	0.03

Figure B.19 in Appendix B.7 and Figure 4.5 show the interaction of these paired differences with key structural properties. Interestingly, there is a clear trend across the urgency spectrum: as the average reaction time increases, the advantage of 5-minute optimization

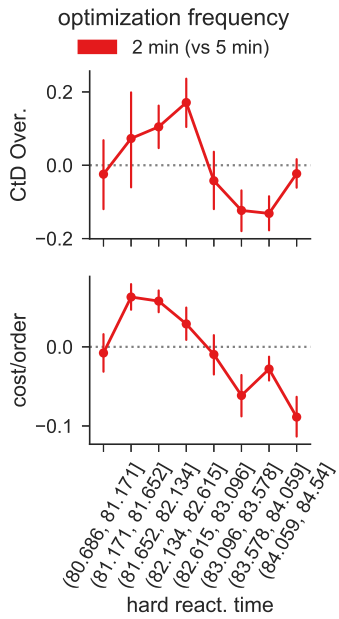


Figure 4.5: Performance difference of algorithm with more frequent optimizations (2 minutes, as opposed to default 5 minutes) vs instance characteristics

intervals evaporates. For instances falling in the upper half of the range of reaction times, *i.e.*, for less urgent scenarios, a 2-minute optimization interval even achieves lower costs per order, as well as lower click-to-door overage and 90th percentile. In our view, as urgency decreases, service time windows for orders become longer, and the space of feasible bundles at any point in time grows larger, which may erode the effectiveness of the insertion heuristics used to select bundles. Shrinking the optimization period (and the assignment horizon in the same proportion) reduces the space of feasible bundles, thus preserving the effectiveness of the heuristics in less urgent environments.

Furthermore, as the ratio of orders to courier hours increases, the default algorithm tends to have relatively more orders per bundle than the 2-minute interval alternative, reflecting the greater ability to consolidate orders of an algorithm with less frequent assignments, which allows the default algorithm to maintain an advantage, albeit diminishing, in terms of click-to-door (mean, 90th percentile and overage), and ready-to-pickup (mean

Table 4.11: Differences in performance of algorithm with longer assignment horizon (20 min) vs default (10 min)

	Assignment horizon				paired differences	
	10 minutes		20 minutes		(u = 20) - (u = 10)	
	mean	std	mean	std	mean	std
% undelivered	0.25	0.46	0.22	0.51	-0.02	0.17
CtoD mean	32.74	3.34	33.15	3.37	0.42	0.55
CtoD 90%	49.3	5.6	50.43	6.01	1.13	1.5
CtoD overage	2.77	1.53	3.11	1.65	0.34	0.38
RtoP mean	1.93	1.52	2.05	1.45	0.12	0.41
RtoP 90%	6.04	4.24	6.87	4.83	0.83	1.57
utilization mean	0.63	0.12	0.66	0.12	0.03	0.01
utilization 10%	0.31	0.15	0.34	0.18	0.03	0.06
cost per order	11.42	1.26	11.41	1.34	-0.01	0.19
orders per bundle	1.09	0.05	1.13	0.06	0.04	0.02

and 90th percentile). However, the trend is steady enough that, in the upper third of the range of this ratio, 2-minute optimization intervals lead to lower percentage of orders undelivered *and* lower cost per order. These results highlight the fact that, in practical applications, this important parameter must be tuned.

Impact of horizon for assignments. Figures B.20 in Appendix B.7 and 4.6, as well as Table 4.11, summarize the performance difference between variation 4, which considers orders for assignment if they are ready within 20 minutes, and the default algorithm, where the assignment horizon is 10 minutes.

A longer assignment horizon leads to more orders per bundle and higher courier utilization levels. Meanwhile the default assignment horizon is preferable in terms of click-to-door and ready-to-pickup, though this advantage comes at the price of a slightly higher percentage of orders undelivered. The difference in service performance tends to become smaller for instances with low urgency, less orders placed, and low order to courier hours ratio, but its sign persists across almost all levels of all instance characteristics, with two clear exceptions: (i) the instances with low dynamism and very poor quality schedules

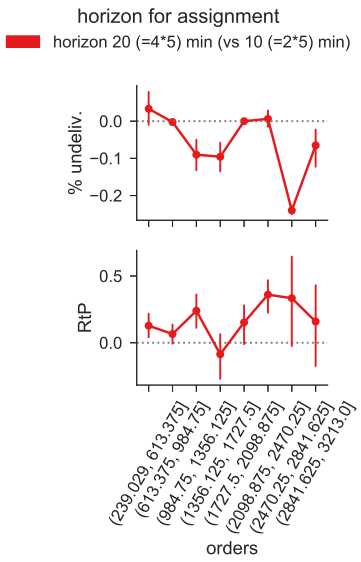


Figure 4.6: Performance difference of algorithm with a 20 minute horizon for assignment of orders (as opposed to default 10 minute horizon) vs instance characteristics

summarized in Table 4.9, where the default algorithm gives up some quality of service in order to get a higher percentage of orders delivered; and (ii) a group of instances with high dispersion and high urgency, summarized in Table 4.12, where the default algorithm struggles to create enough bundles, sacrificing ready-to-pickup (and some order deliveries altogether) in order to preserve the rest of performance metrics, while variation 4 achieves the same feat with better ready-to-pickup (and more orders delivered) by virtue of its reduced myopism.

In terms of cost per order, the default assignment horizon tends to yield better results in instances with low urgency, while the opposite is true for instances with high urgency. As a tentative explanation, consider that for a given assignment horizon, higher urgency implies that each order has the chance to be matched to a courier less times before a final commitment is required: a longer assignment horizon compensates by beginning the assignment process for each order earlier in time.

The default assignment also achieves lower cost than variation 4 for low order to

Table 4.12: Illustrating the potential consequences of excessive myopism in assignment horizon

instance	resp. time	orders	disp.	algo.	% undel.	CtoD mean	RtoP mean	util. mean	$\frac{\text{cost}}{\text{order}}$	$\frac{\text{orders}}{\text{bundle}}$
5o50t100s1p100	81.6	1362	25.8	0	0.5	38.3	5.8	0.8	10.4	1.2
5o50t100s1p125	81.1	1362	25.8	0	1.3	42.7	6.4	0.9	10.3	1.2
7o50t100s1p125	81.2	1606	27	0	0.6	38	4.6	0.8	10.4	1.1
5o100t100s1p100	81.6	2724	26.7	0	0.6	37.1	4.2	0.8	10.8	1.2
5o50t100s1p100	81.6	1362	25.8	4	0.3	38	5.3	0.9	10.2	1.2
5o50t100s1p125	81.1	1362	25.8	4	1.0	42.2	5.7	0.9	10	1.2
7o50t100s1p125	81.2	1606	27	4	0.3	36.7	3.1	0.9	10.2	1.1
5o100t100s1p100	81.6	2724	26.7	4	0.3	37.1	3.7	0.8	10.5	1.3

courier hour ratio, but the tables are turned for higher order to courier hour ratio: when there are many orders per courier and bundling becomes critical, a longer assignment horizon allows higher consolidation levels, thus driving cost down.

Impact of myopic look-aheads to define dynamic target bundle sizes. Table 4.13 and Figures 4.7 and B.21 (in Appendix B.7) summarize the performance difference of experiment runs that use different lookaheads to calculate "system intensity" ratio of orders/couriers, versus the default algorithm which computes the ratio based on orders ready and couriers available within the next 10 minutes.

Results confirm that the definition of the dynamic targets has the intuitively expected impact in the overall behavior of the algorithm: the higher the courier lookahead and the smaller the order lookahead, the larger the target sizes tend to be, and the more orders per bundle in the solution.

We note that the percentage of orders undelivered is not sensitive to the configuration of bundling lookaheads: orders are left undelivered when they are not included in enough matchings to find an assignment, but the dynamic target bundle size only influences the number of bundles to create out of a given set of individual orders, and not how such set

Table 4.13: Differences in performance of algorithm with with varying lookaheads for calculation of bundle target sizes (vs. default, which uses 10 minutes for orders and 10 minutes for couriers)

	dynamic target bundle size lookaheads								paired differences					
	10/10		10/20		20/10		20/20		(10/20)-(10/10)		(20/10)-(10/10)		(20/20)-(10/10)	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
% undeliv.	0.25	0.46	0.25	0.45	0.24	0.53	0.25	0.47	0.01	0.12	0.00	0.18	0.01	0.09
CtoD mean	32.74	3.34	32.75	3.36	32.83	3.37	32.81	3.4	0.01	0.18	0.10	0.27	0.07	0.23
CtoD 90%	49.3	5.6	49.27	5.59	49.57	5.8	49.48	5.79	-0.02	0.40	0.27	0.81	0.18	0.61
CtoD overage	2.77	1.53	2.77	1.53	2.84	1.57	2.81	1.58	0.00	0.10	0.07	0.19	0.05	0.16
RtoP mean	1.93	1.52	1.98	1.62	1.87	1.45	1.94	1.57	0.05	0.20	-0.06	0.22	0.01	0.19
RtoP 90%	6.04	4.24	6.22	4.47	5.95	4.09	6.09	4.38	0.18	0.56	-0.09	0.70	0.05	0.68
util. mean	0.63	0.12	0.63	0.12	0.63	0.12	0.63	0.12	0.00	0.00	0.00	0.01	0.00	0.01
util. 10%	0.31	0.15	0.31	0.15	0.31	0.14	0.31	0.14	0.00	0.02	0.00	0.03	0.00	0.02
cost per order	11.42	1.26	11.41	1.26	11.43	1.25	11.42	1.25	-0.01	0.03	0.01	0.06	0.00	0.05
$\frac{\text{orders}}{\text{bundle}}$	1.09	0.05	1.09	0.05	1.11	0.07	1.1	0.06	0.00	0.01	0.02	0.02	0.01	0.01

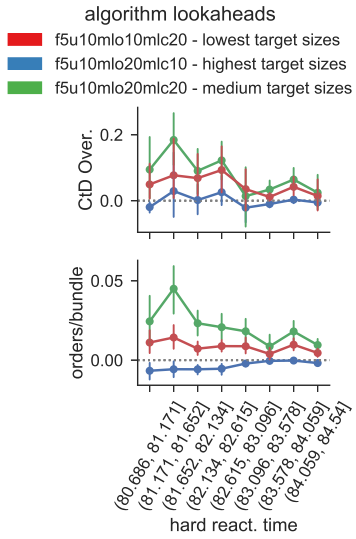


Figure 4.7: Performance difference of algorithm setting alternative lookaheads for bundle targets (default is 10 minutes for both orders and couriers) vs instance characteristics.

is formed. Furthermore, departing from the default configuration rarely improves click-to-door measures (and only slightly, on average), but it can significantly deteriorate these measures in scenarios exhibiting high urgency, or a high ratio of orders to courier hours.

We identify two situations where the configuration with the highest target sizes can lead to significant performance improvements: (i) instances with very little pickup flexibility, as those exemplified in Table 4.14, where it is often beneficial to ramp up bundling in anticipation of bursts of activity during which there will be greater demand and little flexibility; (ii) or instances with capacity problems (historical courier schedules with a high order to courier hour ratio) belonging to the `r50` reduction (*i.e.*, with fewer and more disperse restaurants than full size instances, but more consolidation opportunities than `o50` instances), as those exemplified in Table 4.15, where bundling aggressively whenever possible helps better utilize the scarce courier hours.

Impact of commitment policy. Table 4.16, Figure 4.8 and Figure B.22 (in Appendix B.7) summarize the performance difference between algorithm variation 16, which uses a

Table 4.14: Instances with very low pickup flexibility often benefit from aggressive bundle size targets

instance	pickup flex.	algo.	% undel.	CtoD mean	CtoD 90%	CtoD overage	RtoP mean	RtoP 90%	util. 90%	$\frac{\text{orders}}{\text{bundle}}$
0r50t100s1p125	11.5	0	0	36.8	53	3.7	2.2	7	0.3	1.1
4o100t100s1p125	11.2	0	0	39.4	57	5.6	2.9	9	0.5	1.2
4o50t100s1p125	11.2	0	0.3	43.6	66	8.3	6.8	20	0.5	1.2
4r50t100s1p125	10.5	0	0.2	43.4	66	8.3	5.1	14	0.5	1.2
0r50t100s1p125	11.5	2	0	36.6	53	3.7	1.9	6.9	0.4	1.1
4o100t100s1p125	11.2	2	0	39.7	58	5.9	2.6	8	0.5	1.2
4o50t100s1p125	11.2	2	0.3	43.0	65	7.9	5.9	18	0.5	1.2
4r50t100s1p125	10.5	2	0.2	42.6	65	7.8	3.8	11	0.5	1.3

Table 4.15: Instances with bundling opportunities but scarce effective capacity may benefit from higher dynamic bundle size targets.

instance	$\frac{\text{orders}}{\text{cour. h.}}$	algo.	% undel.	CtoD mean	RtoP mean	util. 10%	$\frac{\text{cost}}{\text{order}}$
1r50t100s1p100	2.1	0	1.1	35.2	5.6	0.4	10.5
1r50t100s1p125	2.1	0	1.9	40.3	6.4	0.4	10.4
7r50t100s1p100	2.3	0	0.1	32.3	3.9	0.3	10.8
7r50t100s1p125	2.3	0	0.3	35.7	3.8	0.3	10.7
1r50t100s1p100	2.1	2	0.8	34.0	4.4	0.4	10.5
1r50t100s1p125	2.1	2	0.0	39.4	5.6	0.4	10.6
7r50t100s1p100	2.3	2	0.1	32.3	3.7	0.4	10.8
7r50t100s1p125	2.3	2	0.3	35.7	3.4	0.3	10.8

single-stage lazy commitment, and the default algorithm, which uses two-stage additive commitment.

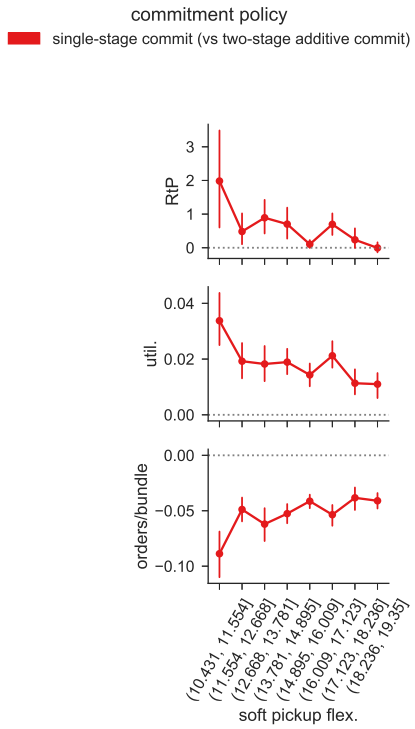


Figure 4.8: Performance difference of algorithm with single-stage commitment rule (as opposed to default two-stage additive rule) vs instance characteristics.

The evidence shows that, compared to the default, a single-stage commitment policy encourages less orders per bundle across the board. Couriers are on the road a longer proportion of time (increased utilization levels), as the algorithm struggles to find solutions that deliver all orders (percentage of orders undelivered doubles) with the same service quality. Meanwhile, ready-to-pickup times degrade by 28% in mean and 20% in 90th percentile, but click-to-door times increase at a much slower pace (about 1% in mean and 90th percentile, and about 5% in overage). We advance a tentative interpretation: if conditions of the system have changed, the two-stage commitment policy has the advantage of allowing bundles to grow in size, or to change their delivery sequence while couriers are en-route or even waiting at the restaurant (thus protecting click-to-door measures, but

Table 4.16: Differences in performance of algorithm with single-stage commitment policy vs. default (two-stage additive commitment)

	Commitment strategy				paired differences (single - two-stage)	
	Two-stage		Single-stage			
	mean	std	mean	std	mean	std
% undelivered	0.25	0.46	0.5	1.29	0.25	1.01
CtoD mean	32.74	3.34	32.95	4.06	0.21	1.11
CtoD 90%	49.3	5.6	49.83	7.61	0.54	3.24
CtoD overage	2.77	1.53	2.91	2.05	0.14	0.79
RtoP mean	1.93	1.52	2.48	2.69	0.55	1.26
RtoP 90%	6.04	4.24	7.29	7.57	1.25	3.79
utilization mean	0.63	0.12	0.65	0.13	0.02	0.02
utilization 10%	0.31	0.15	0.33	0.15	0.02	0.04
cost per order	11.42	1.26	11.36	1.28	-0.06	0.13
orders per bundle	1.09	0.05	1.04	0.02	-0.05	0.03

not ready to pickup measures).

All in all, the results reaffirm the conclusions reached in our reading of Figure 4.4: if the goal is to deliver all orders and deliver them quickly, using of a single-stage commitment policy is not a good idea in any scenario. In fact, the flaws of single-stage commitment are more evident (through a larger increase in the percentage of orders undelivered and all click-to-door and ready-to-pickup measures) on instances with low dynamism, low pickup flexibility, or high ratio of orders to courier hours, that is, as the dynamics of the problem become more challenging.

Impact of priority scheme. Table 4.17, Figure 4.9 and Figure B.23 (in Appendix B.7) summarize the performance difference between algorithm variation 17, which uses a single matching per optimization run, and the default algorithm, which solves a sequence of 3 matchings based on a priority rule for orders and bundles.

For the most part, running the algorithm with or without priority groups does not show much of a difference in terms of orders undelivered, click-to-door measures, or even cost, except for a subset of poorly staffed instances, (as described in Table 4.9), where

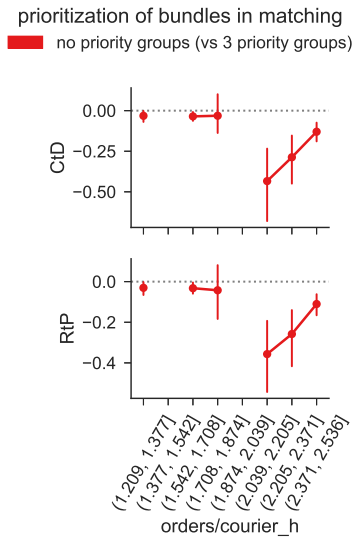


Figure 4.9: Performance difference of algorithm with a single matching problem per optimization run (as opposed to default sequence of 3 matchings based on priority) vs instance characteristics

Table 4.17: Differences in performance of algorithm with no priority scheme to find assignments vs. default, (three priority groups)

	Priority scheme				paired differences (No - Yes)	
	Yes		No			
	mean	std	mean	std	mean	std
% undelivered	0.25	0.46	0.27	0.61	0.03	0.26
CtoD mean	32.74	3.34	32.58	3.16	-0.16	0.42
CtoD 90%	49.30	5.60	49.26	5.55	-0.04	0.92
CtoD overage	2.77	1.53	2.73	1.45	-0.04	0.26
RtoP mean	1.93	1.52	1.79	1.28	-0.14	0.35
RtoP 90%	6.04	4.24	5.59	3.51	-0.45	1.11
utilization mean	0.63	0.12	0.63	0.12	0.00	0.01
utilization 10%	0.31	0.15	0.31	0.14	0.00	0.03
cost per order	11.42	1.26	11.42	1.26	0.00	0.08
orders per bundle	1.09	0.05	1.09	0.05	0.00	0.01

doing away with the priority scheme leads to a significant sacrifice in the percentage of orders undelivered, accompanied by improvements in click-to-door, ready-to-pickup and cost per order: in these instances, during moments when the system is under pressure to keep up with all deliveries, the default algorithm prioritizes delivery of orders that are already in trouble (thereby reducing the number of undelivered orders) and places the burden in orders with lower priority (worsening their click-to-door performance), while variation 17 prioritizes orders that are still on time (to obtain gains in click-to-door times) at the expense of orders that are already late (which sometimes may imply giving up on them altogether). This behavior suggests that on instances with a highly limited supply of couriers, demand management strategies to select which orders to give up on can have a big impact (a topic that will be explored further in Chapter 5).

Impact of bundling. Table 4.18, Figure 4.10, and Figure B.24 (in Appendix B.7) summarize the performance difference of variation 18, where the algorithm is forbidden from bundling orders together (*i.e.* a hard limit of 1 is imposed on the size of bundles), and the default algorithm.

Table 4.18: Differences in performance of algorithm with no bundling allowed vs. default (allow bundling)

	Bundling				paired differences (No - Yes)	
	Yes		No			
	mean	std	mean	std	mean	std
% undelivered	0.25	0.46	0.69	1.77	0.44	1.52
CtoD mean	32.74	3.34	32.72	3.93	-0.02	1.16
CtoD 90%	49.3	5.6	48.79	5.97	-0.50	1.76
CtoD overage	2.77	1.53	2.7	1.74	-0.07	0.63
RtoP mean	1.93	1.52	2.51	2.53	0.58	1.35
RtoP 90%	6.04	4.24	7.35	6.74	1.31	3.42
utilization mean	0.63	0.12	0.66	0.13	0.03	0.02
utilization 10%	0.31	0.15	0.36	0.17	0.05	0.06
cost per order	11.42	1.26	11.26	1.29	-0.15	0.18
orders per bundle	1.09	0.05	1	0	-0.09	0.05

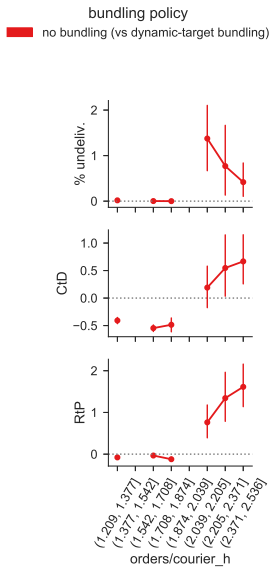


Figure 4.10: Performance difference of algorithm that completely forbids bundling (as opposed to default with dynamic bundling intensity) vs instance characteristics

The results show that allowing bundles is critical for the system: their prohibition almost triples the number of orders undelivered, while deteriorating the ready-to-pickup mean by about 30%, and its 90th percentile by 21%, all for minuscule gains of 1%-2% in click-to-door measures and 1% improvement in cost per order.

Results show that only in the most dynamic, most flexible, less congested and most disperse scenarios can the absence of bundles lead to small improvements in click-to-door without sacrifices in the number of orders delivered. Coincidentally, in these extreme scenarios, the number of orders per bundle used by the default algorithm is smaller, *i.e.*, bundling opportunities are scarce or of little help anyway.

Impact of complexity of the assignment model. Table 4.19, Figure 4.11, and Figure B.25 (in Appendix B.7) summarize the performance difference of experiment runs that depart from the simple matching model, and use larger and more complex assignment procedures.

Overall, results show that the higher complexity assignment model dominates the

Table 4.19: Differences in performance of algorithm with with medium and high complexity assignment models (as opposed to default linear programming model)

	Assignment model complexity						paired differences			
	Low		Medium		High		(Medium - Low)		(High - Low)	
	mean	std	mean	std	mean	std	mean	std	mean	std
% undelivered	0.25	0.46	0.26	0.57	0.25	0.54	0.01	0.22	0.01	0.20
CtoD mean	32.74	3.34	32.72	3.32	32.64	3.34	-0.02	0.19	-0.09	0.22
CtoD 90%	49.3	5.6	49.25	5.6	49.13	5.55	-0.04	0.60	-0.17	0.57
CtoD overage	2.77	1.53	2.76	1.52	2.72	1.52	-0.01	0.10	-0.04	0.13
RtoP mean	1.93	1.52	1.92	1.52	1.91	1.52	-0.01	0.17	-0.02	0.17
RtoP 90%	6.04	4.24	5.98	4.25	5.97	4.21	-0.06	0.72	-0.07	0.74
util. mean	0.63	0.12	0.63	0.12	0.64	0.12	0.00	0.01	0.00	0.01
util. 10%	0.31	0.15	0.31	0.14	0.31	0.14	0.00	0.03	0.00	0.03
cost per order	11.42	1.26	11.42	1.25	11.4	1.25	0.00	0.07	-0.02	0.08
orders per bundle	1.09	0.05	1.09	0.05	1.08	0.05	0.00	0.01	-0.01	0.01

medium complexity model in all performance metrics. And, compared to the default, more complex models trade off a slight increase in orders undelivered for slight improvements in the rest of service performance metrics. We note that the small magnitude of these differences is rather surprising: this suggests that the default approach, while perfectible, goes a really long way in guaranteeing satisfactory performance measures.

At greater detail, it can be noted that both medium and high complexity variations perform similarly in terms of orders delivered, achieving similar values as the default algorithm, except in poorly staffed instances, where Procedure 4 seems to be counterproductive (note that both medium and high complexity versions have this feature in common), as significantly more orders are left undelivered than in the default setting. Closer inspection of the simulation logs reveals that this is caused by a poor decision logic at the end of the operating period: a courier that is about to finish her shift is assigned a follow-up pair, but only commits to the first route; then her shift finishes and the second element of the pair has no courier in a good position to pick up the assignment. This unintended behavior – which may be handled as an exception at the commitment stage in a

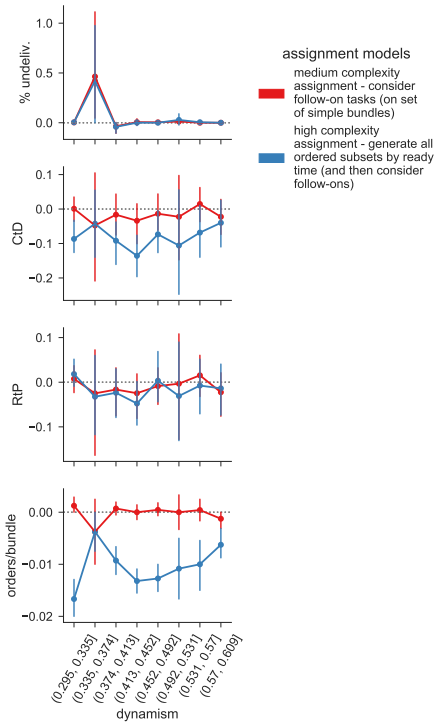


Figure 4.11: Performance difference of algorithm with increasingly complex assignment models (as opposed a matching linear program) vs instance characteristics

production-level application – gives the reader a flavor of the challenges faced during the implementation of fully-automated algorithmic dispatching technologies.

Furthermore, the dominance of the high complexity assignment model over the medium and low complexity models is almost uniform across the space of instance characteristics, except in the case of the very poorly staffed instances. Interestingly, the high complexity model solutions are consistently less bundling intensive than the ones produced by the default algorithm, but they exhibit small yet noticeable improvements in service and cost metrics, in every instance except the same subset of poorly staffed instances.

4.6 Final remarks

In this chapter we have introduced the Meal Delivery Routing Problem, a dynamic deterministic model of the structure and functioning of meal delivery systems, and we have developed a rolling-horizon repeated-matching algorithm to solve this problem in (near) real-time. Our computational results demonstrate that our methodology can be leveraged to build solutions of high quality with respect to multiple performance objectives in meal delivery systems of realistic size and difficulty levels. When confronted with scenarios of high urgency, low flexibility, and scarce capacity, the algorithm makes minimal sacrifices in the rate of fulfillment of deliveries, while sacrificing click-to-door and ready-to-pickup measures in a steady fashion (linear performance degradation in all dimensions).

Moreover, we have shown that the key algorithmic ideas embedded in our method all play an important role: optimization frequency, assignment horizon and commitment policy work together to maintain a reasonable pool of individual orders from which good bundles (and routes) can be made, if needed, without compromising service quality. Meanwhile, the intensity of bundle generation can be modulated effectively using the dynamic target bundle sizes, thus allowing the system to adapt automatically to temporal fluctuations in demand and supply. Furthermore, the proposed priority scheme is critical

to keep up with the pace of arriving orders in the most overwhelming scenarios.

As discussed in Section 4.1, meal delivery and ride-hailing systems share similarities in scale, urgency and dynamism, and our results confirm that a matching approach, which has already shown its merits in ride-hailing, is also effective in meal delivery. While a linear matching model can be very expressive and powerful, achieving high quality solutions fast, careful design of slightly more complex assignment models can correct some of the pitfalls stemming from an excessively myopic solution approach, thus improving overall performance by a small but significant margin, while keeping run-times in check.

The ideas and results reported in this chapter give rise to a variety of questions for further research. In the realm of deterministic models, the development of an exact algorithm for the perfect-information version of the MDRP is clearly a worthwhile endeavor. In a more practical note, restaurant-dependent dynamic target bundle sizes are a natural extension of our approach that would allow the system to compensate for *geographic* imbalances in supply and demand. Furthermore, a systematic study of the value of relocation decisions that are completely decoupled from the order commitment logic is an important question.

In the stochastic realm, opportunities are abundant. While we recognize the importance of predicting order arrival patterns and preparation times, in our opinion, the questions that deserve greater attention are those related to courier autonomy. For instance, independent-contractors usually work for more than one company (sometimes simultaneously), and they leverage their autonomy to reject orders in a strategic way. Developing models to understand assignment rejection behavior, and its relation to system performance, is a fundamental step in the road towards a stochastic dynamic solution algorithm.

Another interesting phenomenon related to courier autonomy is "courier drainage": couriers make waiting and relocation decisions based on incomplete local information

about the system in which they operate. Over time, individual autonomous decisions may lead to too many couriers on areas that are perceived as "profitable" and too few of them in areas that are not perceived as such. The system-wide consequences emerging from courier drainage may not be in the best interest of the central provider, restaurants, and even couriers themselves. Understanding the dynamics of courier drainage, and developing strategies to mitigate it, is an exciting (and daunting) research direction.

CHAPTER 5
DEMAND AND CAPACITY MANAGEMENT IN DYNAMIC DELIVERY

5.1 Introduction

Capacity planning and routing technologies can only go so far in guaranteeing a steady performance in realistic applications. After all, immunizing system performance against remotely possible scenarios likely leads to prohibitively expensive plans. Hence, it is reasonable to assume that a dynamic delivery system will now and then encounter situations to which it cannot adapt quickly. In such situations, demand management interventions can be used to stabilize the system or, at the very least, minimize performance deterioration until appropriate capacity changes (*e.g.*, enroll additional couriers to work some extra hours on a specific region) become feasible.

The use of demand management tactics in urban logistics has been pioneered in recent times by ride-hailing companies. Dynamic pricing schemes like Uber's "surge pricing" or Lyft's "prime time" are able to increase or decrease demand for trips in a certain region at a given time, a feature that may be called *demand smoothing* (of course, rising fares are also intended to rally up more drivers). However, in some contexts of dynamic delivery, like meal delivery, this is not the full picture. Here, an important new feature is present, namely *demand redirection* opportunities: typically, ride-hailing platforms cannot alter the geographic origin or destination of a potential ride (beyond, perhaps, the use of hubs at massive public events); in contrast, it seems reasonable to assume that for a significant proportion of customers, meal delivery platforms may be able to influence the choice of the restaurant (*i.e.*, the origin of the delivery trip) that will fulfill the potential demand for meals like pizzas, chicken, noodles, etc.

Demand management has long been studied in network and queueing research (*e.g.*, queue admission policies, [67, 68, 69, 70]), inventory management (*e.g.*, product substitution, [71]), and revenue management (where "flexible products", closely linked to our concept of "demand redirection", have recently received some attention [72, 73]). However, to the best of our knowledge, the use of demand management strategies in dynamic transportation systems has only begun to be the object of scientific research in the last few years, and mainly in the form of "surge pricing" economic analysis [74, 75, 76].

In contrast to the economic framework that focuses on welfare or utility optimization, the ultimate goal in the research program we initiate in this writing is to understand the effects of demand smoothing and redirection on logistical performance as measured directly by performance indicators like number of orders delivered, click-to-door, ready-to-door, cost-per-delivery, etc. We expect that redirecting and even turning away customers in a strategic way can lead to important improvements in the operation of the system, but, needless to say, doing this has important consequences that go beyond potential short-term revenue losses and whose impact is hard to measure (*e.g.*, reputation, etc.). Nonetheless, assuming that the magnitude of this impact depends on the proportion of customers turned away is a sensible first approximation. Motivated by this reasoning, we begin our exploration of demand management tactics by studying the trade-off between logistical performance and number of customers served (or the number of customers "lost") in simplified dynamic delivery models.

Concretely, in this chapter, we introduce some extensions of the TSP models in Chapter 3, which, in addition to release dates and deadlines, include order selection decisions, enforced in two ways: i) directly controllable selection of individual orders at the time of their release, and ii) indirect selection through the control of a "service coverage radius" that dictates the maximum distance from the depot of any order that will be served, depending only on the order placement time. After developing dynamic programming al-

gorithms to solve the perfect information variant of each problem and establishing some complexity results, we propose heuristic algorithms based on re-optimization for the dynamic problem variants, and conduct simulations to assess their relative performance.

5.2 Routing with release dates, service guarantees, and order selection

We keep all the notation from Chapter 3. As a refresher, $N = \{1, \ldots, n\}$ represents a set of customers located on the real half line $\mathbb{R}^+ \equiv [0, +\infty)$. A single depot is located at $x = 0$. The location of customer $i \in N$ is given by τ_i, and for simplicity we equate travel distances and times, so that delivering to i requires $2\tau_i$ distance and time.

Each customer $i \in N$ places an order at time a_i, with release time r_i (with $a_i \leqslant r_i$), and orders can either be declined (immediately after placement) or delivered by the end of the planning horizon T. When a common service guarantee, S, is enforced, $r_i + S$ is the deadline by which i must be delivered (if i is not declined), and $l_i = r_i + S - \tau_i$ is the latest possible dispatch time of i from the depot.

For a delivery route serving the set of customers $K = \{i_1, i_2, \ldots, i_k\}$, we define the earliest dispatch time as $r(K) = \max\{r_{i_1}, r_{i_2}, \ldots, r_{i_k}\}$, the latest dispatch time as $l(K) = \min\{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$, and the furthest order visited as $\tau(K) = \max\{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_k}\}$ (for convenience, the empty route has $r(\emptyset) = 0$, $\tau(\emptyset) = 0$, $l(\emptyset) = T$).

The first kind of problems that we study are individual order selection problems. We focus on two formulations, with and without a common service guarantee:

Problem 6. *Determine the maximum number of orders, v^*, that can be delivered by a single courier in routes starting and ending at the depot, such that each order i served is dispatched at or after r_i, and the last route returns to the depot by time T.*

Problem 7. *Determine the maximum number of orders, v^*, that can be delivered by a single courier in routes starting and ending at the depot, such that each order i delivered is dispatched*

at or after r_i and delivered before the end of the corresponding service window $r_i + S$, and the last route returns to the depot by time T.

The second kind of problems that we study are "service coverage radius" problems. In this operating environment, instead of making a service acceptance decision for each individual order, demand on the system must be managed only through the control of the service coverage radius, which dictates the maximum distance from the depot of any order that will be served. More precisely, the coverage radius is a mapping of times to distances $\rho : \mathbb{R}^+ \to \mathbb{R}^+$ such that order $i \in N$ must be delivered if and only if $\rho(a_i) \geqslant \tau_i$. In practice, a customer attempting to order a meal would only be allowed to use restaurants at a distance not greater than the respective service coverage radius at the moment of ordering.

The corresponding problems that arise under this operating assumption are:

Problem 8. *Given a set W of predetermined moments at which the coverage radius may change, determine what the radius should be set to in each period, if the objective is to maximize the number of orders, v^*, delivered by a single courier in routes starting and ending at the depot, in such a way that all orders whose distance from the depot does not exceed the service coverage radius at placement time are delivered, and the last route returns to the depot by time T.*

Problem 9. *Given a set W of predetermined times at which the coverage radius may change, determine the optimal radius in each period, if the objective is to maximize the number of orders, v^*, delivered by a single courier in routes starting and ending at the depot, in such a way that all orders whose distance from the depot does not exceed the service coverage radius at placement time are delivered before the end of the corresponding service window $r_i + S$, and the last route returns to the depot by time T*

Remark 1. These four problems all accept the trivial solution where no orders are delivered.

Remark 2. As in Chapter 3, we can restrict our attention to finding optimal solutions with non-interlacing routes: given an arbitrary feasible solution, we can ignore the information about orders declined (*i.e.*, their release times, deadlines and locations are irrelevant once the decision to decline is made), and then use the results from Chapter 3 to transform the solution to one with non-interlacing routes, no later completion time, and no less orders delivered.

5.3 Solving perfect information variants

5.3.1 Individual order selection and no individual service guarantees

We start by developing a dynamic program to solve Problem 6, the case with no individual service guarantees. As usual in this setting, we may assume, without loss of generality, that for $i < j$, we have $\tau_i \geqslant \tau_j$ (other orders can always "piggyback" on a route).

In Klapp [56], an $O(n^2)$ dynamic programming algorithm for the analogue to Problem 6 without order selection is introduced. The algorithm relies on a key structural property that remains valid after introducing order selection (as argued in Remark 2): it suffices to search for optimal solutions to the restricted problem obtained by regarding accept/decline decisions as given.

Remark 3. Concretely, an optimal solution can always be feasibly modified to ensure that:

- the duration of routes strictly decreases with each dispatch
- the courier does not wait at the depot after the first dispatch.

Based on this fact, Problem 6 can be solved using a dynamic program defined by:

- States: (t, b), representing a feasible schedule with first dispatch time at or after $t \leqslant T$, delivering a subset of orders located at a distance strictly smaller than $b \in \mathbb{R}^+ \cup \{\infty\}$ (where $b = \infty$ means that any order, no matter how distant, can be served).

- Values: $v(t,b)$, for $t \geqslant 0$ and $b \in \mathbb{R}^+ \cup \{\infty\}$, counting the maximum number of orders that can be served in a schedule that starts no earlier than t and does not deliver any order whose distance is b or larger.
- Recursion:

$$v(T,b) = 0 \;\; \forall\, b \in \mathbb{R}^+ \cup \{\infty\} \tag{5.1}$$

$$v(t,b) = \begin{cases} \max\left\{ v(r^N(t,\infty),\infty), \max\limits_{d\in\mathbb{R}^+} \left\{ \sum\limits_{\substack{i:\, \tau_i \leqslant d \\ r_i \leqslant t}} 1 + v(r^N(t,d),d) \,\middle|\, \substack{t+2d\leqslant T \\ d\in\{\tau_i : r_i \leqslant t\}} \right\} \right\}, \text{if } b{=}\infty \\ \max\limits_{d\in\mathbb{R}^+} \left\{ \sum\limits_{\substack{\tau_i \leqslant d \\ i:\; r_i \leqslant t \\ r^P(t,b)<r_i}} 1 + v(r^N(t,d),d) \,\middle|\, \substack{d<b \\ t+2d\leqslant T \\ d\in\{\tau_i : r^P(t,b)<r_i\leqslant t\}} \right\}, \text{otherwise} \end{cases}$$

where $r^P(t,b) = \max\limits_{i\in N}\{r_i : r_i \leqslant t-2b\}$ indicates the latest release time of an order that could have been served by a route that covered distance d and completed at time t; $r^N(t,\infty) = \min\limits_{i\in N}\{r_i : r_i > t\}$ is the first order released strictly after time t; and, for $d \in \mathbb{R}^+$, $r^N(t,d) = \min\limits_{i\in N}\{r_i : r_i \geqslant t+2d\}$ (or $r^N(t,d) = T$ if the set is empty) indicates the first order release time after completion of a route starting at t and covering distance d.

The optimal solution to Problem 6 can be found by computing $v^* = v(0,\infty)$.

The recursion can be interpreted as the process of building a schedule backwards from time T, in every step appending one more route to the beginning of the schedule, so that t, previously the start time of the schedule, becomes the completion time of the first route. When b is set to $b = \infty$, no more routes are added to the schedule. Schedules built in this way contain no wait after the very first route is dispatched.

Condition $d < b$, applicable when the route added to the schedule has duration $2b$,

enforces the structure of strictly decreasing route duration on all solutions considered by the algorithm. Condition $t + 2d \leqslant T$ guarantees that the operating period is respected.

Given that the first route in the (t, b) schedule covered distance b, the summation term counts orders released after $r^P(t, b)$ (which could not possibly be served in the first route) but ready by time t and no farther than d, the distance of the farthest order included in the next route (orders with larger distance are declined). In an optimal solution, posterior routes will deliver a total of $v(r^N(t, d), d)$ orders, all of which are released after t and have distance strictly smaller than d.

In summary, the algorithm finds the schedule with the most deliveries among all those that respect the structure outlined in Remark 3. But we know that the optimal schedule in this subset must also be globally optimal.

Proposition 7. *The value $v^* = v(0, \infty)$, can be computed in $\mathcal{O}(n^3)$ time.*

Proof. First note that the initial step in the recursive computation of $v^* = v(0, \infty)$ will automatically lead to evaluating $v(r_1, \infty)$. And after that, all the states visited by the recursion will have t equal to a release time r_i or T. Furthermore, since optimal dispatch routes cover a distance in $\{\tau_i : i \in N\}$ (*i.e.*, it is never optimal to dispatch a route whose farthest point from the depot is not a customer location), the number of states whose value must be computed to determine v^* is $\mathcal{O}(n^2)$.

Then note that sorting all orders by increasing distance once at the beginning of the calculations takes $\mathcal{O}(n \log(n))$ time, but it allows us to compute the immediate reward of all actions out of arbitrary state (t, d) in $\mathcal{O}(n)$ time, by building the summation terms for increasing values of d.

As there are $\mathcal{O}(n^2)$ states to be evaluated and the immediate reward computations take $\mathcal{O}(n)$ time at each state, the whole process takes $\mathcal{O}(n^3)$. □

5.3.2 Individual order selection and no individual service guarantees - alternative approach

While Dynamic Program 5.1 is quite elegant and efficient, it cannot be extended easily to accommodate service guarantees. The dynamic program below does provide a better starting point, albeit that generality comes at the cost of increased computational complexity:

- States: (j, X, t), where "stage" counter $j \in \{0\} \cup N$ tracks the orders $\{0, 1, \ldots, j\}$ so far considered for inclusion in routes; $X \subseteq \{1, \ldots, j\}$ is the subset of orders included in routes; and $t \geqslant 0$ is the completion time of the non-interlacing routes used to deliver X.
- Values: $v(j, X, t)$, for $j \in \{0\} \cup N$, $X \subseteq \{1, \ldots, j\}$, $t \geqslant 0$, counting the number of orders included in routes.
- Recursion:

$$v(0, \emptyset, 0) = 0 \tag{5.2}$$
$$v(j, X, t) = \max \left\{ v(i, X_i, t_i) + |X_{ij}| \,\middle|\, \begin{array}{c} 0 \leqslant i \leqslant j-1 \\ X_{ij} \subseteq \{i+1, \ldots, j\} \\ \max\{t_i, r(X_{ij})\} + 2\tau(X_{ij}) \leqslant t \end{array} \right\}$$

The solution to Problem 6, *i.e.*, a set of routes delivering the maximum number of orders from N during time T, is given by $v^* = \max \{v(n, X, t) | X \subseteq N, t < T\}$. Worryingly, the recursion of Dynamic Program 5.1 seems to suffer from the curse of dimensionality: there are 2^n subsets of N, and, what is worse, there are an exponential number of extensions from state (i, X_i, t_i) to stage j, as we must decide which subset of orders from $\{i+1, \ldots, j\}$ to deliver in a single route departing at or after t_i, and which orders to ignore altogether. But, to our fortune, appearances are in this case misleading, and the recursion can be solved in polynomial time, as we now show.

Proposition 8. *At stage j, there are at most j non-dominated states (j, X, t), one for each possible*

cardinality of X.

Proof. Suppose two partial solutions for stage i lead to states $s_1 = (j, A, a)$ and $s_2 = (j, B, b)$, with $|A| = |B|$, and $a < b$ (W.L.O.G.). Then, any feasible sequence of routes that leads from s_2 to state $s_3 = (n, B + Y, b + c)$, serving a subset of orders $Y \subseteq \{j + 1, \ldots, n\}$, is also feasible starting from s_1: the courier can just wait at the depot between times a and b and then start executing said sequence of routes, finally reaching state $s_4 = (n, A + Y, b + c)$, which completes at the same time as s_3 and delivers as many orders. We conclude that for each possible value of $|X|$ (*i.e.*, for $0, \ldots, j$), there is one state with minimum completion time, which dominates all other states achieving the same number of orders delivered. □

In concordance with Proposition 8, the only extensions from state (i, X_i, t_i) to stage j that may have a chance to be non-dominated are those where routes complete as early as possible. In other words, to efficiently discover non-dominated states in Dynamic Program 5.2, we rely heavily on a sub-problem that deserves attention on its own right:

Problem 10. *Given a set of orders Q, a time t, and a number k, construct a single route that departs no earlier than t, delivers a subset of k orders, and completes as early as possible.*

Proposition 9. *Problem 10 can be solved in quadratic time.*

Proof. Without loss of optimality, we can assume that a route delivering $X \subseteq Q$ will depart at time $\max(t, r(X))$, and therefore have earliest completion time equal to

$$\max(t, r(X)) + 2\tau(X) = \max(t, \max\{r_i | i \in X\}) + 2\max\{\tau_i | i \in X\}$$

Since each order may have a different release time and travel times, there are at most $|Q|^2$ different feasible $(r(X), \tau(X))$ pairs, each determining an earliest completion time.

We now provide Algorithm 7, which solves Problem 10 for all $k \leqslant |Q|$ in tandem, by building a route delivering the maximum number of orders for each feasible $(r(X), \tau(X))$ pair.

To establish the correctness of Algorithm 7, note that each entry $\mathbb{V}_{i,j}$ contains the maximum number of orders in a route starting at $\max(t, r(Y_i))$ and traveling as far as $\tau(Z_j)$. Hence, for any given k:

- if $\mathbb{V}_{i,j} < k$, then no feasible route starting at time $\max(t, r(Y_i))$, and with travel time $\tau(Z_j)$, can deliver k orders.
- and if $\mathbb{V}_{i,j} > k$, a route with release time $r(Y_i)$ and travel time $\tau(Z_j)$ that delivers k orders cannot possibly have minimum completion time: one can always modify route $\mathbb{X}_{i,j}$, by removing either Y_i or Z_j (and more orders with binding release times or travel times, if necessary), to obtain a feasible route with k deliveries that has strictly smaller release time or travel time (or both).

From a complexity standpoint, sorting orders by ready time and travel time requires operations in the order of $\mathcal{O}(|Q| \log(|Q|))$, while $\mathcal{O}(|Q|^2)$ suffice to compute $\mathbb{X}, \mathbb{V}$ and $\{X_k : k \leqslant |Q|\}$. Therefore, the whole solution process takes $\mathcal{O}(|Q|^2)$ time. □

Corollary 10. *All non-dominated states at stage j of the recursion can be found in polynomial time.*

Proof. First, consider the evaluation of extensions from state (i, X_i, t_i) to stage j, when a subset of orders $X_{ij} \subseteq \{i+1, \ldots, j\}$ must be chosen to be delivered in a single route, while the rest of orders are denied service. Algorithm 7 can be used to efficiently find all $(j-i)$ relevant extensions in $\mathcal{O}((j-i)^2)$ time.

Next, note that Proposition 8 guarantees that at stage j there are at most j non-dominated states, but it also implies that there are as many as $j(j-1)/2$ extensions from previous stages that may reach stage j. Hence, finding the states with minimal completion time for

Algorithm 7: Find routes with minimum completion time for each possible cardinality

Input: Q: set of orders
t: earliest start time
Output: $\{X_k : k \leqslant |Q|\}$: Set of routes with minimum completion time, for each k
$Y \leftarrow$ sequence of orders in Q sorted by increasing ready time
$Z \leftarrow$ sequence of orders in Q sorted by increasing travel time
$\mathbb{X} \leftarrow |Q| \times |Q|$ empty array to record orders in a route of maximum size for each (r, τ) pair
$\mathbb{V} \leftarrow |Q| \times |Q|$ empty array to record the size of routes in $\mathbb{X}$
`/* First, discard` (r, τ) `pairs if order related to` τ `is released after` r `*/`
for $i \leftarrow 1, \ldots, |Q|$ **do**
  $j \leftarrow$ position of order Z_i in Y
  $h \leftarrow 1$
  while $h < j$ **do**
    $\mathbb{X}_{i,h} \leftarrow \emptyset$, $\mathbb{V}_{i,h} \leftarrow 0$
    $h \leftarrow h + 1$
`/* Then, evaluate remaining entries of` $\mathbb{X}$ `and` $\mathbb{V}$ `*/`
for $j \leftarrow 1, \ldots, |Q|$ **do**
  $i \leftarrow$ position of order Y_j in Z
  $h \leftarrow 0$
  $B_0 \leftarrow \emptyset$ `// orders ready by` $r(Y_j)$ `with travel times no larger than` $\tau(Z_h)$
  for $h \leftarrow 1, \ldots, |Q|$ **do**
    if $\mathbb{V}_{h,j} = 0$ **then** $B_h \leftarrow B_{h-1}$
    else
      $B_h \leftarrow B_{h-1} \cup \{Z_h\}$
      if $h < i$ **then**
        $\mathbb{X}_{h,j} \leftarrow \emptyset$, $\mathbb{V}_{h,j} \leftarrow 0$
      else
        $\mathbb{X}_{h,j} \leftarrow B_h$, $\mathbb{V}_{h,j} \leftarrow |B|$
`/* Finally, for each` $k \leqslant |Q|$`, find route in` $\mathbb{X}$ `with minimum completion time */`
for $k \leftarrow 1, \ldots, |Q|$ **do**
  $X_k \leftarrow \arg_{\mathbb{X}_{i,j} \in \mathbb{X}} \min \{\max(t, r(\mathbb{X}_{i,j})) + 2\tau(\mathbb{X}_{i,j}) : \mathbb{V}_{i,j} = k\}$
return $\{X_k : k \leqslant |Q|\}$

each possible cardinality takes some time, $\mathcal{O}(j)$, to be precise (achieved by doing a single pass through the set of all states in stage j, while keeping j tabs of the running minimum for each cardinality). Furthermore, the time needed to compute all extensions to stage j is

$$A_j = \mathcal{O}(1(j-1)^2) + \mathcal{O}(2(j-2)^2) + \ldots + \mathcal{O}(j(1)^2) = \mathcal{O}\left(\frac{(j-1)j^2(2j-1)}{12}\right) = \mathcal{O}(j^4)$$

□

Since the recursion of Dynamic Program 5.2 must traverse n stages, it follows that

Corollary 11. *Solution* $v^* = \max\{v(n, X, t) | X \subseteq N, t < T\}$ *can be found in* $\mathcal{O}(n^5)$ *time.*

5.3.3 Individual order selection with individual service guarantees

We now introduce service guarantees, and formulate a dynamic program to solve Problem 7, based on the insights discussed in the previous section:

- States: (j, X, t), where "stage" counter $j \in \{0\} \cup N$ tracks the orders $\{0, 1, \ldots, j\}$ so far considered for inclusion in routes; $X \subseteq \{1, \ldots, j\}$ is the subset of orders included in routes; and $t \geqslant 0$ is the completion time of the non-interlacing routes used to deliver X.
- Values: $v(j, X, t)$, for $j \in \{0\} \cup N$, $X \subseteq \{1, \ldots, j\}$, $t \geqslant 0$, counting the number of orders included in routes.
- Recursion:

$$\begin{aligned} v(0, \emptyset, 0) &= 0 \\ v(j, X, t) &= \max\left\{ v(i, X_i, t_i) + |X_{ij}| \,\middle|\, \begin{array}{c} 0 \leqslant i \leqslant j-1, \\ X_{ij} \subseteq \{i+1, \ldots, j\}, \\ \max\{t_i, r(X_{ij})\} + 2\tau(X_{ij}) \leqslant \min\{t, l(X_{ij})\} \end{array} \right\} \end{aligned} \tag{5.3}$$

The solution to Problem 7 can be found by computing $v^* = \max\{v(n, X, t) | X \subseteq N, t < T\}$, and solving the recursion still takes polynomial time: Proposition 8 holds (and its proof

needs no modifications to deal with deadlines), while Proposition 9 also holds, but Algorithm 7, used in the proof to solve Problem 10 efficiently, must be slightly modified to deal with deadlines.

As a consequence of the additional deadline restrictions, which are now enforced by Algorithm 8, it is possible that Problem 10 has no solution for some $k \leqslant |Q|$, *i.e.*, there are no routes delivering k orders. In the algorithm, this is reflected in the fact that the maximum value of entries in $\mathbb{V}$ may be strictly smaller than $|Q|$. Nonetheless, these changes do not affect the arguments proving correctness of Algorithm 8, and its complexity is again quadratic.

We conclude that, following the same scheme as in Section 5.3.2, Dynamic Program 5.3 can still be solved in $\mathcal{O}(n^5)$ time.

5.3.4 Coverage radius problems with and without service guarantees

To solve problems 8 and 9 we can follow a simple enumeration strategy, and rest assured that for any given set of radius change times W, the optimal solution to each problem can be computed in time exponential in $|W|$ only, as shown below.

Proposition 12. *Given a set of radius change times W, the optimal radius profile for an instance with n orders is one among $\mathcal{O}(n^{|W|})$ relevant radius profile alternatives.*

Proof. Given a set W of times when the service coverage radius can change, there are $|W|$ decision epochs (*i.e.*, intervals between consecutive radius change times), each with as many relevant radius alternatives as orders placed within it (one for each order distance value, as there is nothing to gain by setting the radius to a value that is not exactly the distance of some order).

For an arbitrary instance with n orders, suppose there are $q_1, \ldots, q_{|W|}$ orders placed within each of the corresponding decision epochs, with $\sum_{j}^{|W|} q_j = n$. Given the number of

Algorithm 8: Find feasible routes with minimum completion time for each possible cardinality in the presence of service guarantees

Input: Q: set of orders. t: earliest start time
Output: $\{X_k : k \leqslant |Q|\}$: Set of routes with minimum completion time, for each k
$Y \leftarrow$ sequence of orders in Q sorted by increasing ready time
$Z \leftarrow$ sequence of orders in Q sorted by increasing travel time
$\mathbb{X} \leftarrow |Q| \times |Q|$ empty array to record orders in a route of maximum size for each (r, τ) pair
$\mathbb{V} \leftarrow |Q| \times |Q|$ empty array to record the size of routes in $\mathbb{X}$
`/* First, discard` (r, τ) `pairs if order related to` τ `is ready after` r`, or must be dispatched before` $\max(t, r)$ `*/`
for $i \leftarrow 1, \ldots, |Q|$ **do**
  $j \leftarrow$ position of order Z_i in Y
  $h \leftarrow 1$
  while $h < j$ **do**
    $\mathbb{X}_{i,h} \leftarrow \emptyset$, $\mathbb{V}_{i,h} \leftarrow 0$
    $h \leftarrow h + 1$
  $a \leftarrow \min\{\, b : j \leqslant b \leqslant |Q|,\ l(Z_i) < \max(t, r(Y_b))\}$
  for $h \leftarrow a, \ldots, |Q|$ **do**
    $\mathbb{X}_{i,h} \leftarrow \emptyset$, $\mathbb{V}_{i,h} \leftarrow 0$
`/* Then, evaluate remaining entries of` $\mathbb{X}$ `and` $\mathbb{V}$ `*/`
for $j \leftarrow 1, \ldots, |Q|$ **do**
  $i \leftarrow$ position of order Y_j in Z
  $h \leftarrow 0$
  $B_0 \leftarrow \emptyset$ `//` $r(Y_j)$ `with travel times no larger than` $\tau(Z_h)$
  for $h \leftarrow 1, \ldots, |Q|$ **do**
    if $\mathbb{V}_{h,j} = 0$ **then** $B_h \leftarrow B_{h-1}$
    else
      $B_h \leftarrow B_{h-1} \cup \{Z_h\}$
      if $h < i$ **then**
        $\mathbb{X}_{h,j} \leftarrow \emptyset$, $\mathbb{V}_{h,j} \leftarrow 0$
      else
        $\mathbb{X}_{h,j} \leftarrow B_h$, $\mathbb{V}_{h,j} \leftarrow |B|$
`/* Finally, for each` $k \leqslant |Q|$`, find route in` $\mathbb{X}$ `with minimum completion time */`
for $k \leftarrow 1, \ldots, |Q|$ **do**
  $X_k \leftarrow \arg_{\mathbb{X}_{i,j} \in \mathbb{X}} \min \{\max(t, r(\mathbb{X}_{i,j})) + 2\tau(\mathbb{X}_{i,j}) : \mathbb{V}_{i,j} = k\}$
return $\{X_k : k \leqslant |Q|\}$
.

orders in each period, in the worst case, there are $(1+q_1)(1+q_2)\ldots(1+q_{|W|})$ interesting radius configurations. This quantity is bounded above by $(1+\frac{n}{|W|})^{|W|}$ (due to the AM-GM inequality). Therefore, if $|W|$ is fixed, this function is $\mathcal{O}(n^{|W|})$. □

Since order selection is completely determined once the value of the radius at each change point has been set, to evaluate each radius profile, it suffices to solve the corresponding feasibility problem from Chapter 3, with or without service guarantees, which can be done in $\mathcal{O}(n^2)$ in either case. While this enumerative procedure may very well not be the most efficient, it is correct because alternatives are evaluated exhaustively.

Corollary 13. *Problems 8 and 9 can be solved in $\mathcal{O}(n^{|W|+2})$ time.*

It is worth noting that not all conceivable radius profiles have to be fully evaluated, as bounds on the optimal solution can be calculated with relative ease. The corresponding individual order selection problem provides an upper bound u to the number of orders that can be covered by the optimal radius profile (so, profiles that cover more orders than u need not be considered). Meanwhile, feasible solutions with non-decreasing objective value can be obtained by restricting the radius change points to nested subsets of W.

5.4 Dynamic problem variants

We now proceed to propose algorithms to solve the on-line version of the problems introduced in Section 5.2. We explore two approaches, one based on "roll-out" policies of the off-line individual selection algorithms, and another, more simple approach, based on periodic service coverage radius changes. After describing the approaches, we compare their relative performance in a simple simulation experiment with varying instance size, service guarantee, distances and preparation times.

As stated earlier, a critical constraint in our dynamic environment is that orders can only be declined at placement time. Additionally, in the heuristics proposed in this sec-

tion, we will assume that routing decisions are final once all orders are released and the courier is available at the depot.

5.4.1 Roll-out of individual order selection algorithms

Dynamic Programs 5.1 and 5.3 can be deployed in a dynamic environment using the roll-out logic outlined in Algorithm 9. In theory, the worst-case computational burden grows by a factor of $O(n)$ compared to the off-line, perfect information, counterpart. In practice, because orders are not typically placed all at the same time, the size of the problems solved when each order arrives is much smaller than n, and the on-line algorithm runs faster than its perfect information counterpart.

5.4.2 Dynamic service coverage radius

When managing the service coverage radius in a dynamic environment, there is always the risk that the service coverage radius selected for a given period turns out to be too large after the fact, making it impossible to deliver all orders within the radius on time, even without individual service deadlines. In other words, unlike in the perfect-information environment, additional "recourse" logic to make service decisions for orders placed within the coverage radius is necessary to guarantee feasibility of the operations.

One simple way to solve this is to decline every order that lies within the coverage radius but whose acceptance would lead to a schedule with completion time later than a given limit. Said limit may simply be T, for any period of the radius profile (*i.e.*, , all remaining orders are rejected, and the radius becomes 0 at later change points), but this may lead to solutions where too many early accepted orders occupying courier time that would be more productive serving orders that arrived later. Alternatively, to prevent early over-commitment, the limits may be defined as a function of the change times: given any pair w_k, w_{k+1} of consecutive change times in W, time limit $h_k = w_{k+1} + f$ would imply

Algorithm 9: Dynamic roll-out of DP 5.1 or DP 5.3

Input: T: time horizon. Q: set of orders.
Output: $\mathbb{X} = \{X_k\}$ and $\mathbb{D} = \{d_k\}$: Sequence of routes and corresponding dispatch times (for all k, $X_k \subseteq Q$ and $0 \leqslant d_k \leqslant d_k + 1 \leqslant T$)

$t \leftarrow 0$, current time
$A(t) \leftarrow \emptyset$, set of orders announced by time t, accepted, and not yet dispatched
$Y \leftarrow \emptyset$, schedule of tentative routes
$X \leftarrow \emptyset$, first route to be dispatched from Y
$c \leftarrow 0$, completion time of last committed route
$d \leftarrow T$, dispatch time of next tentative route
while $t \leqslant T$ **do**
  Wait for next event time t_e. It may be an order arrival, or a route dispatch at time d.
  if $t_e < d$ **then**
    `// event: order arrival`
    $o \leftarrow$ order just placed
    $Y' \leftarrow$ optimal schedule produced by DP 5.1 for Problem 6 (or DP 5.3, for Problem 7, with service guarantees) defined for orders $A(t) \cup \{o\}$ and start time $\max(c, t_e)$
    if $o \in Y'$ and $q \in Y'$ $\forall q \in A(t)$ **then**
      $A(t_e) \leftarrow A(t) \cup \{o\}$ `// order accepted`
      $Y \leftarrow Y'$
      $X \leftarrow$, first route in updated tentative schedule
      $d \leftarrow \max(c, r(X))$, earliest dispatch time of updated first route
    else
      $A(t_e) \leftarrow A(t)$ `// order declined`
  else
    `// event: route dispatch`
    $\mathbb{X} \leftarrow \mathbb{X} \cup \{X\}$
    $\mathbb{D} \leftarrow \mathbb{D} \cup \{d\}$
    $A(t_e) \leftarrow A(t) \setminus X$
    $Y \leftarrow Y \setminus X$
    $c \leftarrow d + 2\tau(X)$
    $X \leftarrow$ first route in updated tentative schedule, if any
    $d \leftarrow \max(c, r(X))$ if $X \neq \emptyset$, otherwise T
  $t \leftarrow t_e$

that an order arriving in period $(w_k, w_{k+1}]$ and covered by ρ_k is declined if any schedule serving all accepted orders and the new order finishes after h_k. For reasonably small f (*e.g.*, of magnitude comparable to the average or maximum preparation time), this limit effectively reserves courier capacity for the orders that may arrive in future radius change periods.

Having motivated the need for a "recourse" mechanism, we now outline the approach in Algorithm 10. The key motivation behind it is that if the order arrival process does not change drastically between consecutive periods, the history of the last period can be used to evaluate the radius decision for the next period (provided that the consecutive periods have approximately the same duration).

5.5 Simulation experiments

5.5.1 Setup

To gain insights into the practical performance of the approaches studied, we have designed a simple simulation experiment. Instances with varying numbers of customers ($n = 80, 100, 120, 140$) on the half-line have uniform random distances from the depot, in the range $(0, R]$, ($R = 40, 50, 60$). Preparation times are sampled from a uniform random distribution in the range $(0, P]$ ($P = 10, 15, 20$). Order placement times follow a uniform distribution over a fixed time horizon ($T = 840$ minutes, *e.g.*, 10 am to midnight). In case a service guarantee is made, promising delivery within S minutes since the ready time, S can take values $S = R, R + 5, R + 10$. For each combination of (n, R, P, S), we have obtained 20 instantiations, adding up to a grand total of 2160 instances.

In this way, we are able to control the intensity of the arrival process (directly related to n, as T is invariant), dispersion of locations (all else equal, larger R increases the expected distance of orders from the depot), degree of myopism in a dynamic environment (all

Algorithm 10: Dynamic radius management algorithm

Input: T: time horizon. Q: set of orders. ρ_0: initial service radius.
$W = \{w_1, w_2, \ldots, w_k\}$: increasing sequence of radius change times.
$f \geqslant 0$: buffer after radius period ends for completion of routes.

Output: $\mathbb{X} = \{X_k\}$ and $\mathbb{D} = \{d_k\}$: Sequence of routes and corresponding dispatch times (for all k, $X_k \subseteq Q$ and $0 \leqslant d_k \leqslant d_k + 1 \leqslant T$)

$t \leftarrow 0$, current time
$A(t) \leftarrow \emptyset$, set of orders announced by time t, accepted, and not yet dispatched
$Y, X \leftarrow \emptyset, \emptyset$, schedule of tentative routes, and first route to be dispatched from it
$c \leftarrow 0$, completion time of last executed route
$d \leftarrow T$, dispatch time of next tentative route
$j \leftarrow 0$, current radius change period
while $t \leqslant T$ **do**
  Wait for next event time t_e. It may be an order arrival, a radius change time, or a route dispatch (at time d).
  if $t_e < \min(d, w_{j+1})$ **then**
    $o \leftarrow$ order just placed
    if $\tau_o > \rho_j$ **then**
      $A(t_e) \leftarrow A(t)$ `// order outside service coverage radius`
    $Y' \leftarrow$ schedule serving all of $A(t) \cup \{o\}$, starting at $\max(c, t_e)$, ending as early as possible
    if $\min(T, w_{j+1} + f) < c(Y')$ **then**
      $A(t_e) \leftarrow A(t)$ `// order declined (to avoid over-commitment if` $c(Y') < \infty$`, or deadline conflicts if` $c(Y') = \infty$`)`
    else
      $A(t_e) \leftarrow A(t) \cup \{o\}$ `// order accepted`
      $Y \leftarrow Y'$
      $X \leftarrow$, first route in updated tentative schedule
      $d \leftarrow \max(c, r(X))$, earliest dispatch time of updated first route
  else if $d < w_{j+1}$ **then**
    $\mathbb{X}, \mathbb{D} \leftarrow \mathbb{X} \cup \{X\}, \mathbb{D} \cup \{d\}$ `// event: route dispatch`
    $A(t_e), Y \leftarrow A(t) \setminus X, Y \setminus X$
    $c \leftarrow d + 2\tau(X)$
    $X \leftarrow$ first route in updated tentative schedule, if any
    $d \leftarrow \max(c, r(X))$ if $X \neq \emptyset$, otherwise T
  else
    Solve Problem 8 (or 9) for orders placed in period $(w_j, w_{j+1}]$ and a courier available at time $w_j + \max(0, c - w_{j+1})$, to determine the ρ^* that would have led to the most deliveries complete before $\min(w_{j+1} + f, T)$.
    Set $\rho_{j+1} = \rho^*$.
  $t \leftarrow t_e$

else equal, larger P tends to increase the amount of information about future dispatches that can be used to make acceptance/rejection decisions), and flexibility of the system in the presence of individual service guarantees (all else equal, larger values of S tend to increase the number of feasible solutions).

In the case of perfect information radius management algorithms, we experiment with 3 variants: with a single change point ($W = \{0\}$), 2 change points ($W = \{0, \frac{T}{2}\}$), and 4 change points (at $W = \{0, \frac{T}{4}, \frac{T}{2}, \frac{3T}{4}\}$). We exploit the nested structure of the sets to recycle solutions as lower bounds and speed up the solution process.

Algorithm 10 requires the definition of parameters f and ρ_0, in addition to the sets of change-points ($\{0\}$, $\{0, \frac{T}{2}\}$, $\{0, \frac{T}{4}, \frac{T}{2}, \frac{3T}{4}\}$. We set these auxiliary parameters to default values of $f = T$ and $\rho_o = R$. Finally, we explore the sensitivity of the dynamic radius management algorithm with 4 change points, running variations with $f = 0, S, T$ and $\rho_0 = R, \frac{R}{2}$.

5.5.2 Results

We begin by summarizing the average percentage of orders delivered by each algorithm, in Table 5.1, for the environment with perfect information, and Table 5.2, for the dynamic environment.

Table 5.1: Average performance of perfect information algorithms for coverage maximization

% Orders delivered	Service guarantee No	Yes
Individual order selection	91.6	59.5
radius management - $\|W\| = 1$	35.8	34.1
radius management - $\|W\| = 2$	68.1	43.8
radius management - $\|W\| = 4$	84.2	49.9

Remarkably, under perfect information, a few more change points can provide an important boost to the effectiveness of a service coverage radius scheme, particularly when

there are no individual service guarantees: going from one to four times when the radius may be modified more than doubles the number of orders that can be served.

Note also that the performance advantage of the no-service guarantee variant grows with the number of change points. Conversely, if the system is constrained to operate with a static coverage radius, maximizing the number of orders delivered seems to be compatible with offering a service guarantee.

Table 5.2: Average performance of on-line algorithms for coverage maximization.

% Orders delivered				Service guarantee No	Service guarantee Yes
individual order selection				89.9	47.2
	W	f	ρ		
radius management	1	$\emptyset$	R	90.0	43.9
	2	T	R	89.7	43.0
	4	0	$R/2$	24.3	17.0
	4	0	R	28.1	14.2
	4	S	$R/2$	53.9	43.2
	4	S	R	61.4	41.3
	4	T	$R/2$	75.7	48.0
	4	T	R	88.5	44.8

The simulations with a dynamic environment deliver a big surprise: the policy of enforcing a single coverage radius over the operating period - accepting all early orders within the radius until saturating the schedule, to then "call it a day" and decline most orders arriving late in the operating horizon (unless they can be inserted on a route without throwing off the schedule completion time beyond T) - is actually competitive with more complex roll-out algorithms.

Remarkably, when there are no individual service guarantees, the simplest policy beats all other algorithms at their default values. This illustrates that increasing the number of change points can actually be counterproductive if done carelessly

Comparing dynamic radius management algorithms against their perfect information

analogues, we find that the recourse action that allows declining orders that arrive later in the period even if they are within the radius (necessary to guarantee feasibility) brings about an unexpected performance boost, so much that dynamic algorithms on average achieve more deliveries than what their perfect information counterparts ever could without using individual selection.

All of the above results suggest that, if no service guarantees are present, simple heuristics that do not discriminate orders by distance may be more effective than radius management. However, at least on the half-line geometry, the room for improvement is quite limited: compared to the perfect information individual selection algorithm, which provides an upper bound on performance, we note that the value of perfect information - the performance difference between the dynamic algorithm and the perfect information individual order selection algorithm (91.6% of orders covered, in no service guarantee scenario) - is on average quite small for the best performing heuristics of each class.

Enforcing a service guarantee for accepted orders forces the system to take a big hit in absolute performance: the best algorithm is forced to decline more than half of the incoming orders, as opposed to 10% when there are no commitments other than delivery by the end of the horizon. The simple fixed radius policy no longer dominates the rest, and now the additional flexibility in radius management strategies does pay off, as setting 4 change points and $\rho_0 = \frac{R}{2}$ yields the best performance (even beating the individual order selection roll-out algorithm) by allowing the system to decline some of the most distant and disproportionately inflexible orders. Save for the case of the greedy fixed radius strategy, the value of recourse is no longer enough to overcome the dynamic information handicap.

Simultaneously, Table 5.2 shows that the performance radius management scheme is quite sensitive to f and ρ_0. Larger values of f lead to better performance with or without service guarantees, which is consistent with emulating the greedy behavior of the simple

policy. Meanwhile, if there are no service guarantees, setting ρ_0 as large as possible is on average better (consistent with our observation that the recourse decisions create most of the value in this scenario). But if there are service guarantees, setting ρ_0 to a smaller value improves performance in the first period, attesting to the importance of setting the right coverage radius at all times to preserve flexibility in the system.

To complete our analysis, we delve deeper to explore the dependencies between performance and controlled instance characteristics.

Figure 5.1: Legend for Figures 5.2-5.5

Figures 5.2-5.5 summarize the interaction between the objective value (normalized vs n) and instance characteristics like maximum preparation time (P), minimum pickup flexibility ($(S-R)$), order arrival intensity (n is used, since T is constant for all instances), and maximum distance (R). The legend to interpret them is shown in Figure 5.1.

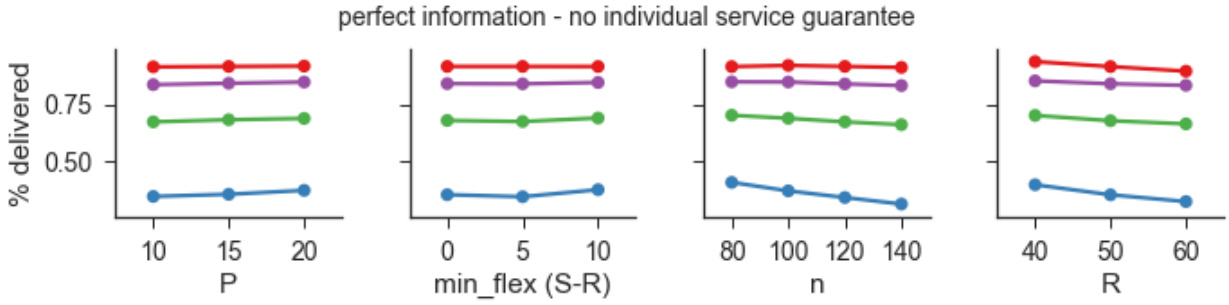


Figure 5.2: Percentage of orders delivered vs instance characteristics, in a perfect information environment with no service guarantees.

There are two discernible trends illustrated in Figure 5.2 for the environment with

perfect information and no service guarantees. First of all, performance in more inflexible radius management schemes deteriorates as orders per unit of time rise, while the individual selection scheme is immune to this change. Second, performance of all schemes decreases as the maximum distance increases, but, again, the effect is stronger in less flexible radius management schemes. P and $(S - R)$ do not have much influence in performance, which is not surprising, given the assumptions of perfect information (which rules out anticipatory benefits of long preparation times) and no service guarantees (so the average pickup flexibility has little to do with $(S - R)$).

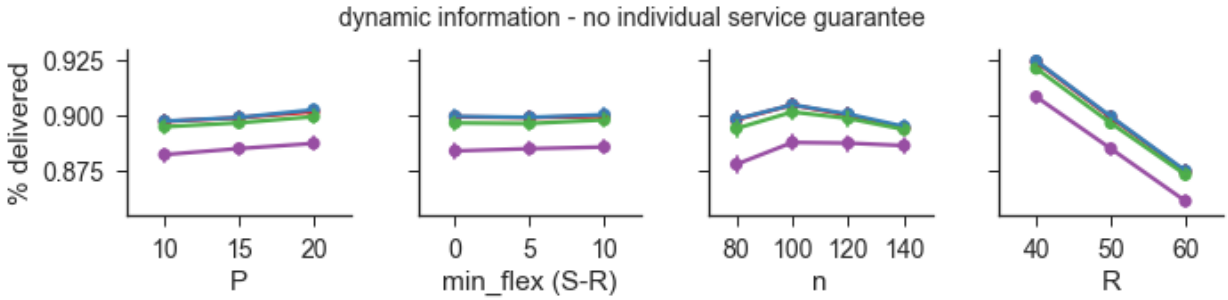


Figure 5.3: Percentage of orders delivered vs instance characteristics, in a dynamic environment with no service guarantees

Figure 5.3 illustrates the performance of dynamic algorithms with no service guarantees. Note that the red line is still being drawn but it is covered by the blue line, illustrating that, save for counted exceptions, the individual selection roll-out yields dispatch decisions comparable (often identical) to the greedy policy of accepting orders as long as they can be packed into a schedule of duration no larger than T.

Interestingly, in a dynamic environment, there seems to be an optimal instance size (round 100 orders), beyond which routes created without complete knowledge of the future cannot become much more productive, thus leading to a decline in the proportion of orders delivered. Also remarkable, although not unexpected, is the sensitivity of the algorithms to the value of R, reflecting the fact that dispatching sub-optimal routes has

a smaller detrimental effect if the duration of said routes is constrained to be smaller. Finally, Figure 5.3 illustrates the small but positive performance effect of anticipation, particularly noticeable for the greedy fixed radius strategy.

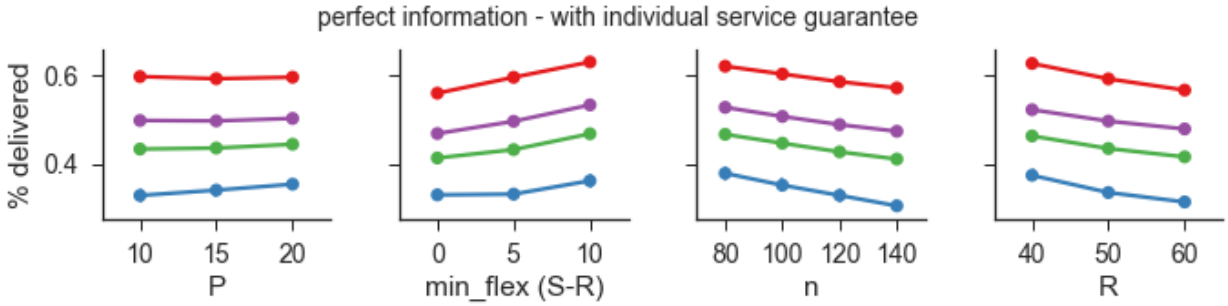


Figure 5.4: Percentage of orders delivered vs instance characteristics, in a perfect information environment with service guarantees

Algorithmic performance under perfect information in the presence of individual service guarantees, summarized in Figure 5.4, follows trends similar to the case with no such guarantees, but in a more dramatic way. As expected, minimum flexibility, $(S - R)$, now has a sizable positive influence on performance: as dispatch time windows grow, so does the space of feasible dispatches, and, at optimality, solutions can serve more orders.

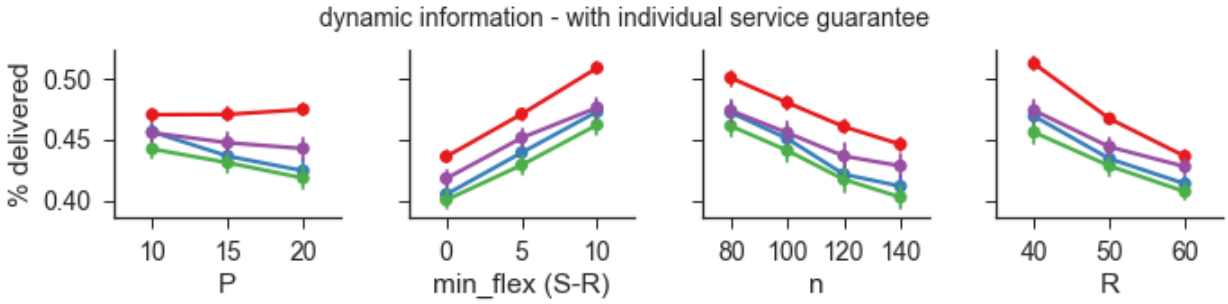


Figure 5.5: Percentage of orders delivered vs instance characteristics, in a dynamic environment with service guarantees

Performance in the dynamic environment with service guarantees is illustrated in Figure 5.5. Here, the individual selection algorithm does lead to significantly more orders delivered, and its relative advantage over the simple greedy policy seems to be indepen-

dent of pickup flexibility and order arrival frequency. It is worth observing how a larger service area erodes the advantage of individual order selection over radius management:

Performance as a function of preparation times exhibits a perplexing behavior: the longer the time difference between announcement and ready times, the worse the radius management algorithms perform. This is true even for the simplest variant with a fixed radius, which suggests that the main reason behind this behavior lies in the nature of the use of a completion time minimization procedure to make acceptance decisions. We advance the following conjecture: the coverage maximization and completion time minimization objectives are not perfectly aligned, and the more advanced information is available (which manifests itself in problems with more orders each time that a decision is made) the more the solutions emerging from the use of each objective may differ.

5.6 Final remarks

The polynomial time individual order selection algorithms developed in this chapter constitute an important tool that can be used in the analysis of demand management interventions of varying degrees of sophistication (*e.g.*, system congestion / late delivery warnings, enforcement of minimum purchase policies, etc.), using models on the simple geometry of the half-line or beyond it (*e.g.*, star geometries).

The existence of more efficient algorithms to solve Problems 7, 8 and 9 remains as an open question, the study of which will hopefully provide additional insights. Additionally, a natural next step is to investigate the complexity of some closely related order selection problem variants. For example, the Algorithms for Problems 6 and 4 exploit the fact that one can dispatch routes of decreasing duration as late as possible (given the set of orders to deliver), so it is interesting to investigate whether a version with the hierarchical objective of (maximum) number of orders delivered and (minimal) total distance traveled can be solved efficiently.

On a similar note, given the small gap in average performance of individual order selection algorithms in dynamic (Algorithm 9) and perfect information settings (Dynamic Program 5.2) in the absence of individual service guarantees, competitive analysis of online Algorithms for Problem 6 (in lines similar to *e.g.*, [77, 78]) could be illuminating.

Furthermore, by methodological decision, the dynamic algorithms introduced in this chapter have not made use of stochastic optimization techniques, but there is clearly room for them, specially in environments with service guarantees, where the choice of initial parameters is critical.

All in all, the theoretical and experimental results provided in this chapter are a small but necessary step on the path to better understand demand management from a logistical perspective, and we hope that the ideas presented here will stimulate more interest in the topic.

Appendices

APPENDIX A

APPENDICES TO CHAPTER 2

A.1 Instance generation

Algorithm 11: GENERATING GENERAL INSTANCES

Input:
n, number of customers; m, maximum number of locations per customer
s, constant speed of all vehicles; T, planning horizon
p, travel distance control ($0 < p \leq 1$)
$avgD$, mean demand; $stdD$, standard deviation of demand
Output:
a set of customer profiles with a demand quantity and a sequence of locations and associated time windows (first and last location are home)
foreach *customer* $c \leftarrow 1, \ldots, n$ **do**
  $d_c \leftarrow$ Random sample from normal($avgD, stdD$)
  $home_c \leftarrow$ Random point in a circle with radius $sT/2$ and center at the origin
  $m_c \leftarrow$ Random integer in $\{1, \ldots, m\}$
  if $m_c = 1$ **then** `// customer has a single location`
    $locsequence_c \leftarrow (home_c)$
    $timewindows_c \leftarrow ([0, T])$
  else `// customer has multiple locations`
    $dailylocs_c \leftarrow$ Random points in circle of radius $sTp/2m$ and center at $home_c$
    $locsequence_c \leftarrow (home_c, dailylocs_c, home_c)$
    $timewindows_c \leftarrow$ Randomly partition the time available among all locations, where time available is T minus the total travel time required to visit $dailylocs_c$

A.2 Algorithm performance details

Here, we delve deeper into the experiment results for the general instances and provide statistics on the contribution of the two problem-specific algorithmic ideas incorporated in the construction and improvement algorithms, i.e., employing the dynamic program

to optimize the locations visited for a given sequence of customers during a switch of neighborhoods, and maintaining a number of alternative schedules to be considered during insertion and deletion operations. In Table A.1, for the two neighborhoods employed during the improvement phase of HEURRDL we present the number of times they were active when a schedule improvement was found as well as the number of times the dynamic program improved the schedule when switching from one neighborhood to the other.

In Table A.2, we present the fraction of insertion and deletion operations in which the flexibility to switch to an alternative location was exploited. We further distinguish between whether the switch to another location was exploited for the customer preceding or succeeding the customer being inserted or deleted.

Algorithm 12: GENERATING REALISTIC INSTANCES

Input:
n, n_2, n_3 : number of customers, and percent of customers with 2 and 3 locations
W_1 : percent of customers working 4 hours and work start $\in$ {8 am, noon}
W_2 : percent of customers working 4-7 hours and work start $\in U$[8 am,10 am]
WC : set of work cluster center coordinates
K_2, S_2 : number and minimum separation of locations in each work cluster
K_3, S_3 : number and minimum separation of after-work locations
p_1, p_2, p_3 : maximum proportion of time a customer can travel 1) before work, 2) after work (total), 3) after work (direct to home distance)
T, T_1, T_2, T_3 : planning horizon and radii of homes, work, and after-work clusters
$avgD, stdD$: mean and standard deviation of demand
Output: a set of customer profiles with a demand quantity and a sequence of locations and associated time windows (first and last location are home)

foreach $c \leftarrow 1, \ldots, n$ **do**
  $m_c \leftarrow$ number of locations that customer c will have, according to n, n_2, n_3
foreach $c \leftarrow RandomOrder(1, \ldots, n)$ **do**
  $ws_c, we_c \leftarrow$ start and end of work time window, according to W_1, W_2.
$L_1, L_2, L_3 \leftarrow \emptyset, \emptyset, \emptyset$ sets of potential home, work and after-work locations
$L_1 \leftarrow$ points in circular area of radius T_1 centered at origin
foreach *wc in WC* **do**
  $L_2 \leftarrow L_2 \cup \{K_2$ points within T_2 of wc and with S_2 minimum separation $\}$
$L_3 \leftarrow \{K_3$ points within T_3 from origin and with S_3 minimum separation$\}$
foreach $c \leftarrow 1, \ldots, n$ **do**
  $d_c \leftarrow$ Random sample from normal$(avgD, stdD)$
  if $m_c = 1$ **then** `// customer has a single location`
    $home_c \leftarrow$ uniform-random sample point from L_1
    $locsequence_c, timewindows_c \leftarrow (home_c), ([0, T])$
  else if $m_c = 2$ **then** `// customer has 2 locations`
    $w_c \leftarrow$ random sample from L_2
    $h_c \leftarrow$ uniform-random sample from L_1 and close enough to w_c, given p_1, p_2
    $locsequence_c \leftarrow (h_c, w_c, h_c)$
    $timewindows_c \leftarrow ([0, ws_c - tt(h_c, w_c)], [ws_c, we_c], [we_c + tt(w_c, h_c), T])$
  else `// customer has 3 locations`
    $w_c \leftarrow$ random sample from L_2
    $h_c \leftarrow$ uniform-random sample from L_1 and close enough to w_c, given p_1, p_2
    $a_c \leftarrow$ random sample from L_3 close enough to w_c and h_c, given p_2, p_3.
    $as_c \leftarrow we_c + tt(w_c, a_c)$
    $x \leftarrow$ uniform-random in $[0, (T - we_c - tt(w_c, a_c) - tt(a_c, h_c))]$
    $locsequence_c \leftarrow (h_c, w_c, a_c, h_c)$
    $timewindows_c \leftarrow$
      $([0, ws_c - tt(h_c, w_c)], [ws_c, we_c], [as_c, as_c + x], [as_c + x + tt(a_c, h_c), T])$

Table A.1: Number of successes for the improvement neighborhoods employed by HEUR-RDL.

Instance	RDGR	Switch (DP)	rGDrGR
1	8	3	1
2	7	0	0
3	3	0	5
4	4	0	1
5	6	0	0
6	3	0	0
7	9	1	2
8	6	0	8
9	3	0	0
10	2	0	1
11	12	2	2
12	23	0	1
13	13	1	1
14	15	4	1
15	15	1	0
16	14	1	1
17	18	2	2
18	20	2	1
19	25	0	3
20	25	2	1
21	55	5	0
22	60	7	0
23	36	1	0
24	34	2	0
25	31	0	0
26	34	6	0
27	37	3	0
28	46	3	0
29	37	0	0
30	35	1	0
31	64	3	0
32	89	0	0
33	64	2	0
34	83	1	0
35	72	0	0
36	92	2	0
37	86	3	0
38	72	1	0
39	110	2	0
40	69	1	0

Table A.2: Percentage of multi-location switches.

Instance	% alt. pre. ins	% alt. pre. del	% alt. suc. ins	% alt. suc. del
1	0	0	9	9
2	0	6	13	13
3	0	0	0	0
4	5	0	23	29
5	3	0	0	31
6	4	1	1	0
7	19	20	2	2
8	17	19	11	13
9	18	18	10	10
10	6	6	3	0
11	6	6	21	21
12	10	10	22	24
13	7	6	2	3
14	2	4	10	11
15	5	3	2	4
16	9	10	19	20
17	2	9	9	9
18	2	3	14	25
19	10	12	5	6
20	4	4	6	7
21	2	2	2	2
22	1	1	6	9
23	2	3	2	2
24	9	8	4	4
25	1	2	7	9
26	3	4	7	7
27	10	9	5	6
28	3	2	11	19
29	4	6	4	5
30	6	7	1	1
31	2	3	4	5
32	2	3	2	2
33	2	2	3	4
34	3	4	1	1
35	3	3	3	5
36	2	2	2	3
37	1	1	5	5
38	2	3	1	1
39	3	3	2	3
40	4	4	3	2

APPENDIX B

APPENDICES TO CHAPTER 4

B.1 Instance generation - reduction of order and courier sets

Algorithm 13: Indirect p-Reduction of order set through randomized selection of restaurants

Data: p, the percentage of reduction; O, original set of orders; R, original set of restaurants.
Result: O', reduced set of orders; R', reduced set of restaurants.

$O' \leftarrow \emptyset$
$R' \leftarrow \emptyset$
$n' \leftarrow p * |O|$
repeat
  $r \leftarrow$ restaurant sampled from R (uniform-random without replacement)
  $O_r \leftarrow$ orders placed from O at r
  $O' \leftarrow O' \cup O_r$
  $R' \leftarrow R' \cup \{r\}$
until $|O'| \geqslant n'$
if $|O'| > n'$ **then**
  $x \leftarrow |O'| - n'$ `/* excess from target */`
  $y \leftarrow n' - |O' \setminus O_r|$ `/* deficit from target if last restaurant not in sample */`
  if $y \leqslant x$ **then**
    $O' \leftarrow O' \setminus O_r$ `/* backpedal... */`
    $R' \leftarrow R' \setminus \{r\}$
return O',R'

Algorithm 14: p-Reduction of courier set

Data: p, the percentage of reduction.
S, the set of shift start times in current schedule.
$B_s \forall s \in S$, the set of shifts in the original schedule sharing start time $s \in S$.
Result: $C_s \forall s \in S$, the set of shifts in the reduced schedule sharing start time $s \in S$.

for $s \in S$ **do**
  /* Set target number of hours */
  $c'_s \leftarrow p * c(B_s)$ // $c(B_s)$ is total number of courier hours in B_s
  /* Build C_s by sampling one shift at a time */
  $C_s \leftarrow \emptyset$
  repeat
    $b \leftarrow$ shift sampled from B_s (uniform-random without replacement)
    $C_s \leftarrow C_s \cup \{b\}$
  until $c(C_s) \geqslant c'_s$
  if $c(C_s) > c'_s$ **then**
    $b \leftarrow$ shift of maximal duration in C_s
    $b' \leftarrow$ shift starting at time and location of b, lasting $c(b) - (c(C_s) - c'_s)$
    $C_s \leftarrow \{b'\} \cup C_s \setminus \{b\}$
return $C = \bigcup_{s \in S} C_s$

B.2 Description of instance files

Information for an instance is provided in four text files, with lines formatted as follows (in all files the first line contains helpful information on its content and can be ignored):

Listing B.1: Restaurant set

```
Restaurant ID, x, y
```

Listing B.2: Order set

```
Order ID, x, y, placement time, Restaurant ID, ready time
```

Listing B.3: Courier set

```
Courier ID, x, y, on-time, off-time
```

Listing B.4: Time and compensation parameters

```
Travel time multiplier, service time at pickup,
service time at drop-off, target click-to-door time,
maximum click-to-door time, per-order pay, per-hour pay
```

B.3 Use of solution evaluator

A solution to an instance can be summarized by a list of assignments of orders to couriers. A full specification also includes pickup and delivery times for individual orders, as well as the sequence of locations visited and the corresponding departure times for each courier.

Our solution evaluator receives as input three files. The first file contains one line for each pickup and delivery assignment (which represent decisions and corresponding decision times). The format of each line is as follows (in all files the first line is assumed top contain helpful information on its content and will be ignored):

Listing B.5: Assignment information

```
assignment time, pickup time, Courier ID, Order ID, ..., Order ID
```

Note that if prepositioning or assignmend updates are allowed in the operating environment, only final assignments should be recorded in this file.

The second file contains one line per individual order delivered, formatted as follows:

Listing B.6: Order delivery information

```
Order ID, placement time, ready time,
pickup time, delivery time, courier ID
```

The third file contains one line per movement of a courier. Moves corresponding to each courier are presented in blocs and, within each bloc, lines are written following the

sequence in which the movements are executed. In general, each line is formatted as follows:

Listing B.7: Courier dispatch information

```
Courier ID, departure time, origin, destination
```

where ORIGIN is either 0, indicating the courier's on-location, RESTAURANT ID, or ORDER ID, and DESTINATION is either RESTAURANT ID or ORDER ID.

The evaluator checks that the following conditions are satisfied:

1. each order is assigned at most once;
2. assignments are not made before orders are placed;
3. a courier completes all the pickup tasks before the end of his duty period.
4. the pickup time of a bundle is at or after the latest ready time of the orders in the bundle;
5. orders are delivered in the sequence prescribed by the assignment;
6. the sequence of courier movements is feasible (no tele-transporting), and all arrival and departure times are consistent.
7. at the time of the pickup of an order or a bundle, the courier is in the corresponding restaurant.
8. at the time of the drop off of an order, the courier is in the corresponding diner location.

If a given solution satisfies these conditions, *i.e.* it is feasible, the evaluator returns a file with values for the aforementioned performance metrics. If the solution is infeasible, the evaluator returns a file indicating the ID of couriers and orders involved in an infeasible action.

B.4 Parameter Tuning

Before starting the full-scale experiments, reasonable default values for a series of secondary parameters in the algorithm must be found. These include:

1. tolerances for service loss (click-to-door overage) and freshness loss (delay in pickup beyond ready time) before orders escalate in priority;
2. tolerance for freshness loss of individual orders before forcing the commitment of a bundle;
3. coefficient (reward) for utilization in the matching objective;
4. freshness loss coefficient (penalty) in the matching objective;
5. service loss coefficient (penalty) in the insertion cost function.

To tune these parameters, a preliminary experiment is conducted over a stratified sample of the set of instances: for each of the 24 instance variations, 4 instances are selected at random. Then, a set of 32 random parameters configurations is created by repeating the following procedure (inspired in [79]):

1. The tolerances for service loss and freshness loss before orders escalate in priority are set simultaneously by uniform-randomly choosing a pair from

   $$[(10,5),(15,5),(15,10),(20,5),(20,10),(20,15),(25,5),(25,10),(25,15),(25,20)]$$

   The first entry in the pair represents the service loss tolerance, and the second one represents the freshness loss tolerance.
2. The tolerance for freshness loss of individual orders before forcing the commitment of a bundle is set by uniform-randomly choosing a value in $[15,20,25]$.
3. The utilization coefficient in the matching objective is set by sampling the uniform distribution on the interval $[1,10]$.
4. The freshness loss coefficient in the matching objective is set by sampling the uni-

form distribution on the interval $[0, 1]$.

5. The service loss coefficient in the insertion cost function is set by sampling the uniform distribution on the interval $[0, 1]$.

The resulting algorithm configurations are then tested on each instance. Default values are chosen for each of the 24 instance variations, with two goals in mind: virtually zero orders left-over at the end of the planning period, and minimal average click-to-door. Concretely, an algorithm run is considered 'successful' in a particular instance if no more than 0.5% of orders are left unserved, and a parameter configuration is considered 'successful' in a given instance variation sub-sample if the algorithm succeeds in at least 75% of the cases (*i.e.* in 3 out of 4 instances in the tuning sample). The selection rule is then: among the 'successful' parameter configurations choose one that minimizes average click-to-door.

B.5 Algorithm performance on instance variations

The figures below illustrate performance metrics for different instance subsets, as follows:

- courier schedules:
  - historical [`cour_sched_type`=1]
  - optimized [`cour_sched_type`=2]
- travel times:
  - original [`tt_multip=100`]
  - shorter by 25% [`tt_multip=75`]
- preparation times:
  - original [`prep_multip=100`]
  - longer by 25% [`prep_multip=125`]
- size reductions:
  - original [`size_reduction=o100`]
  - sampling 50% of couriers and 50% of order set [`size_reduction=o50`]
  - sampling 50% of couriers and enough restaurants to obtain

    50% of orders [`size_reduction=r50`]

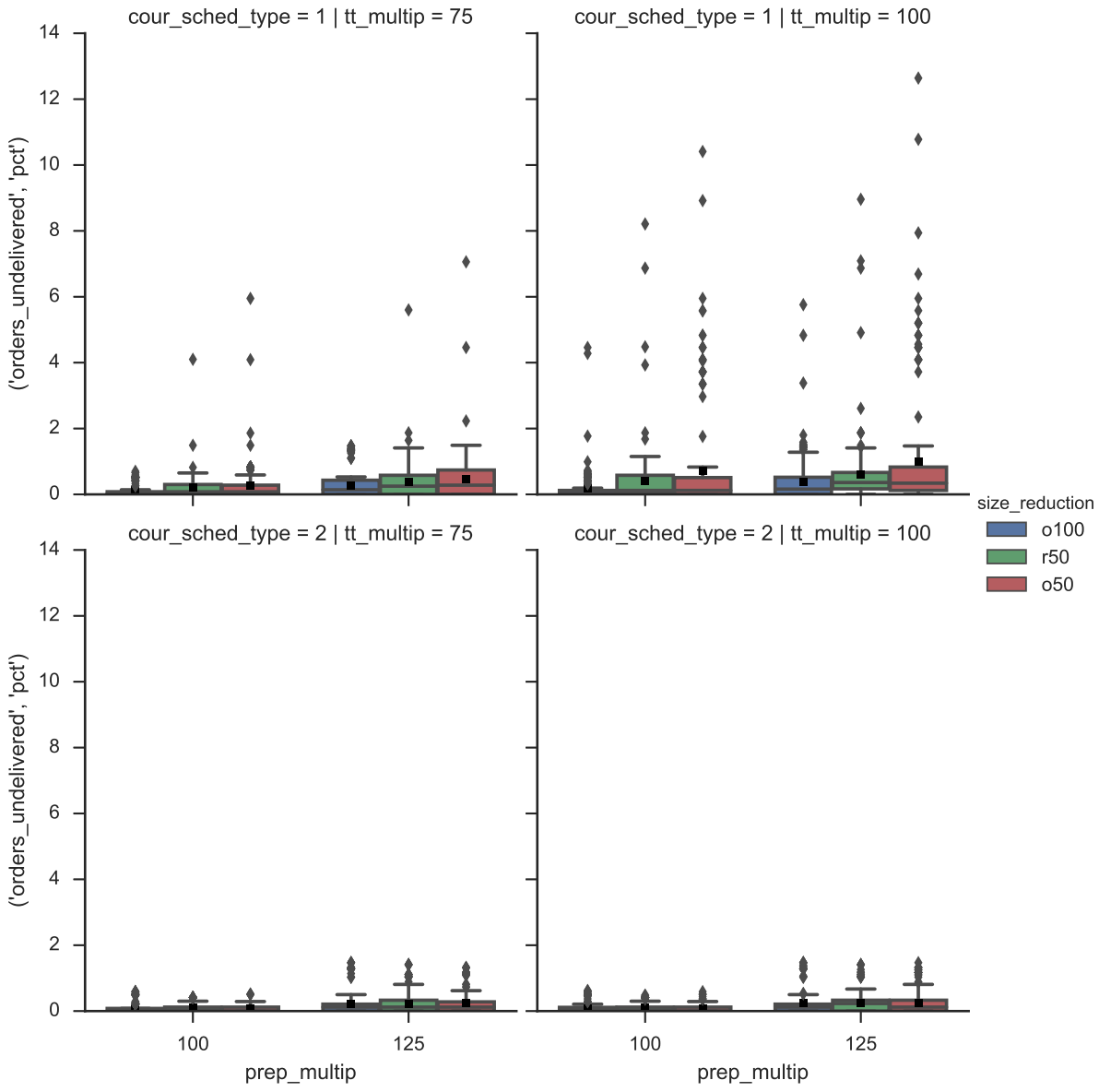


Figure B.1: Percentage of orders undelivered, across different instance classes

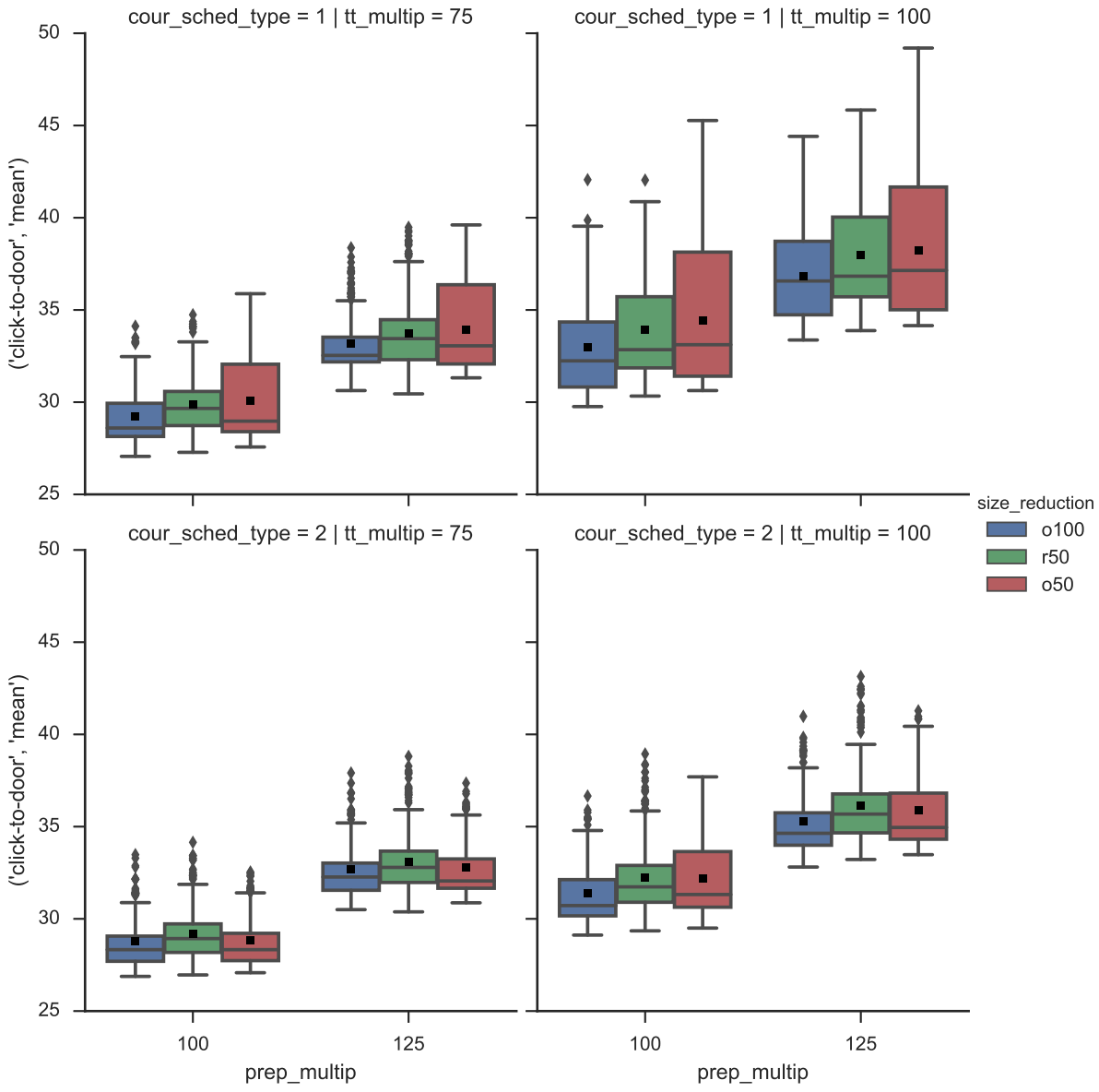


Figure B.2: Click-to-door mean, across different instance classes

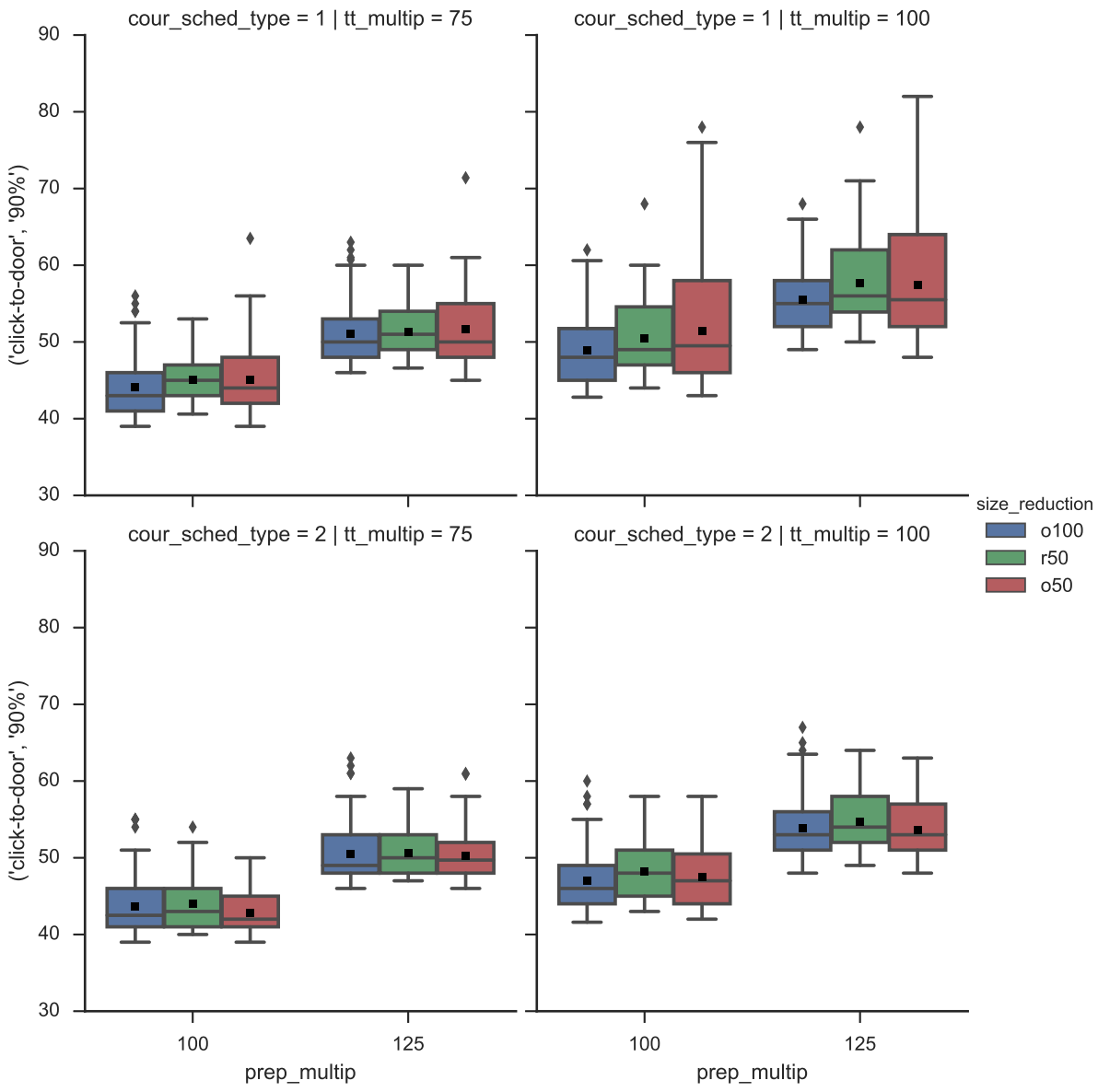


Figure B.3: Click-to-door 90th percentile, across different instance classes

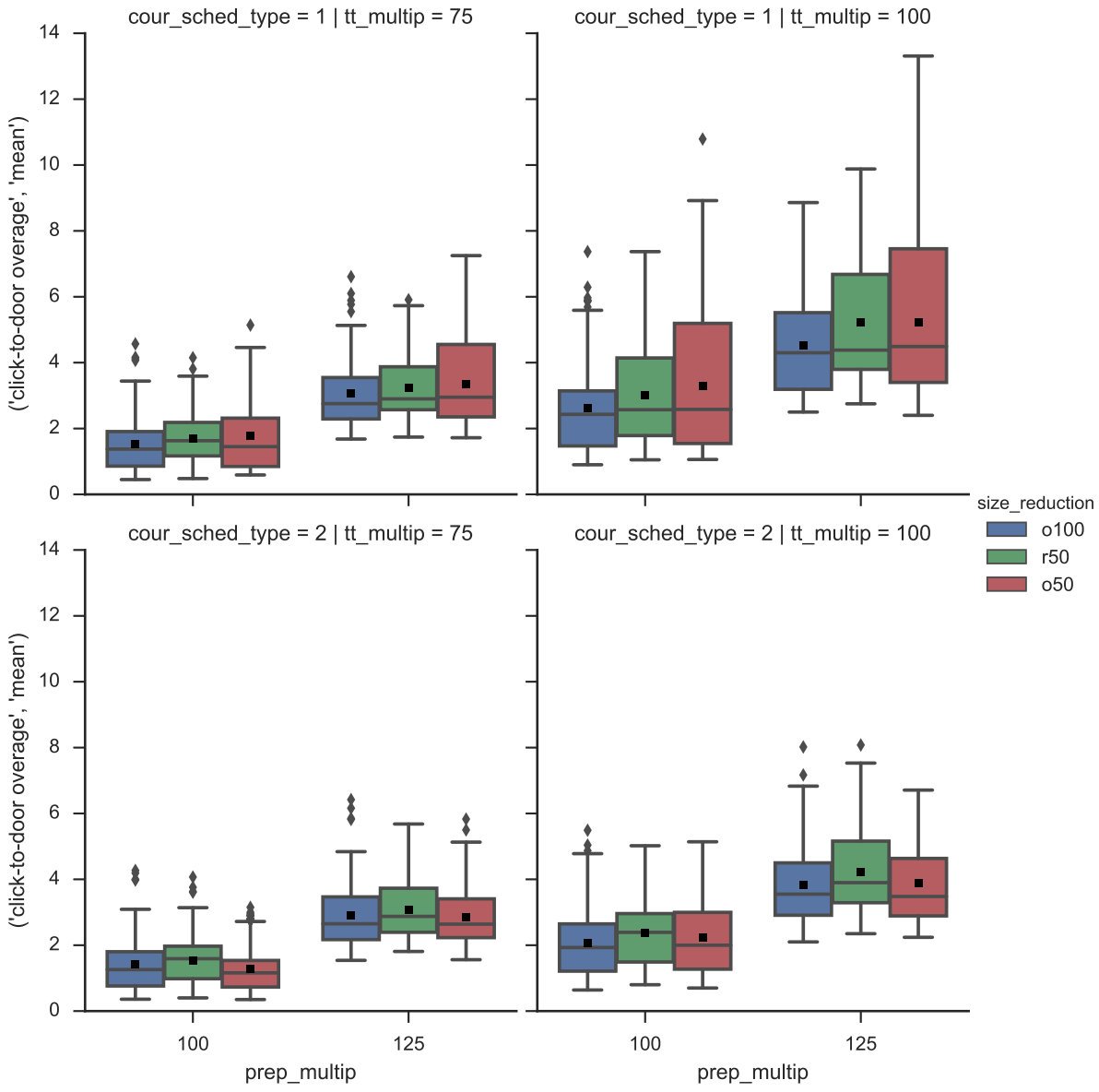


Figure B.4: Click-to-door overage mean, across different instance classes

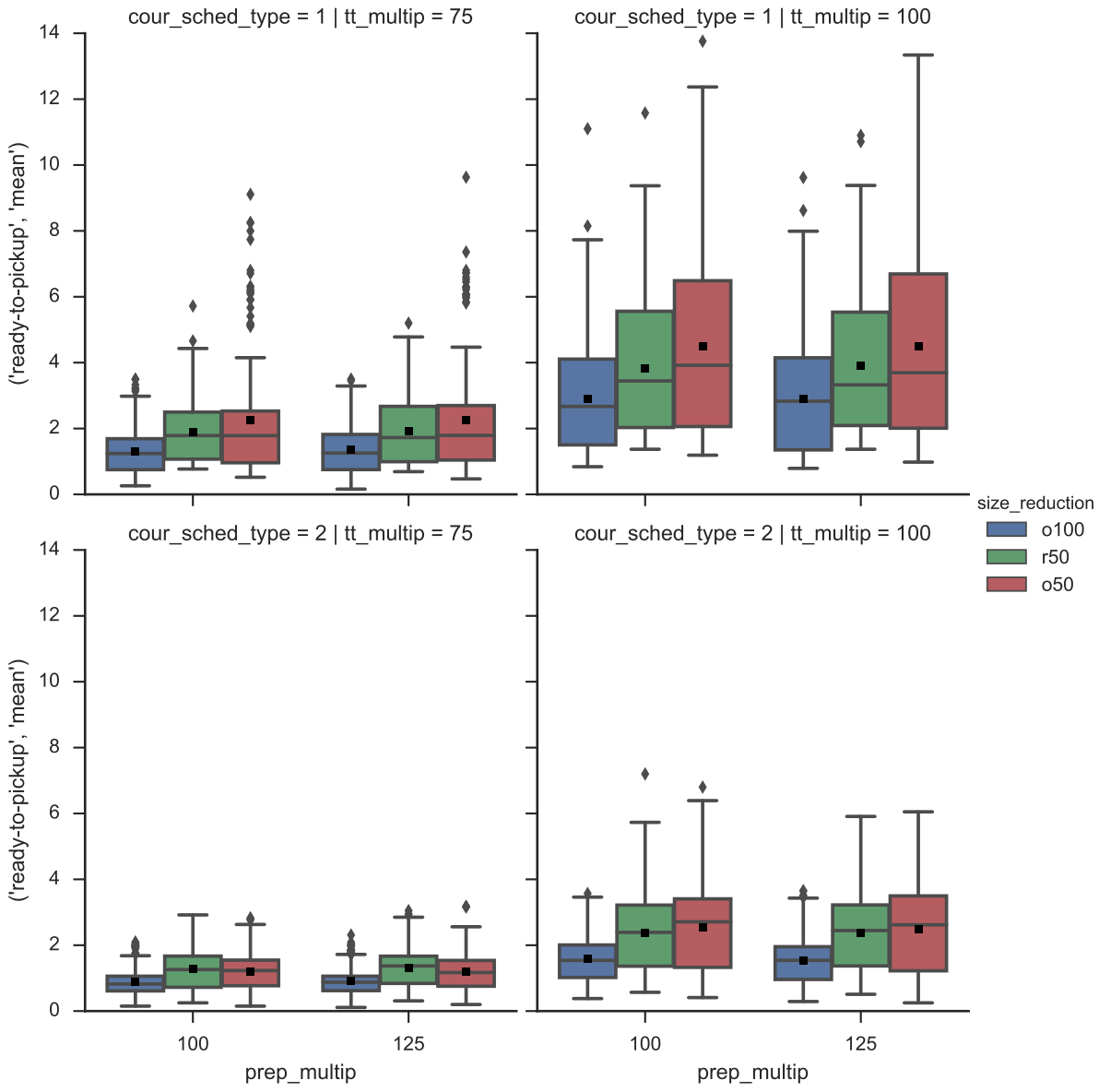


Figure B.5: Ready-to-pickup mean, across different instance classes

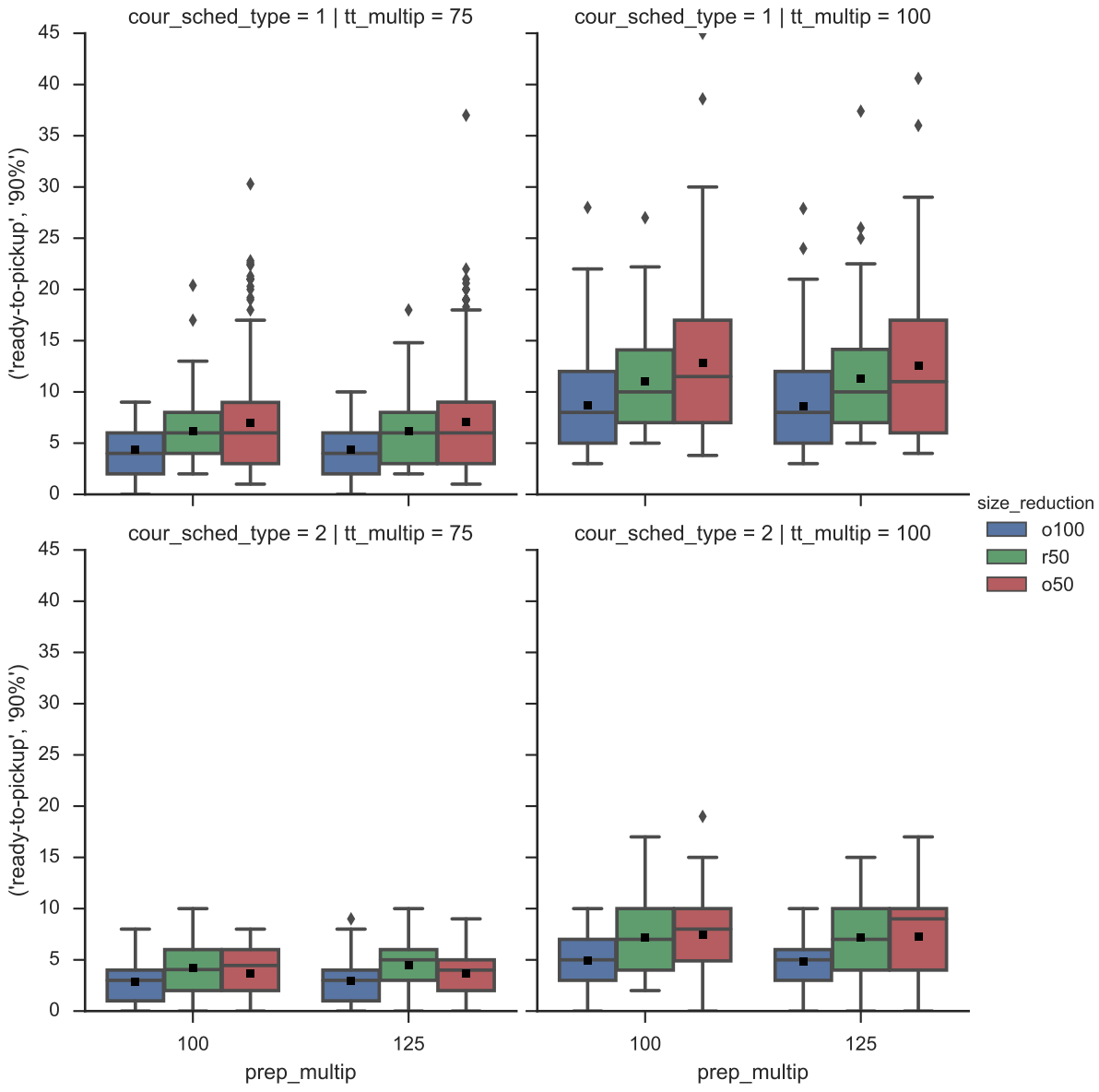


Figure B.6: Ready-to-pickup 90th percentile, across different instance classes

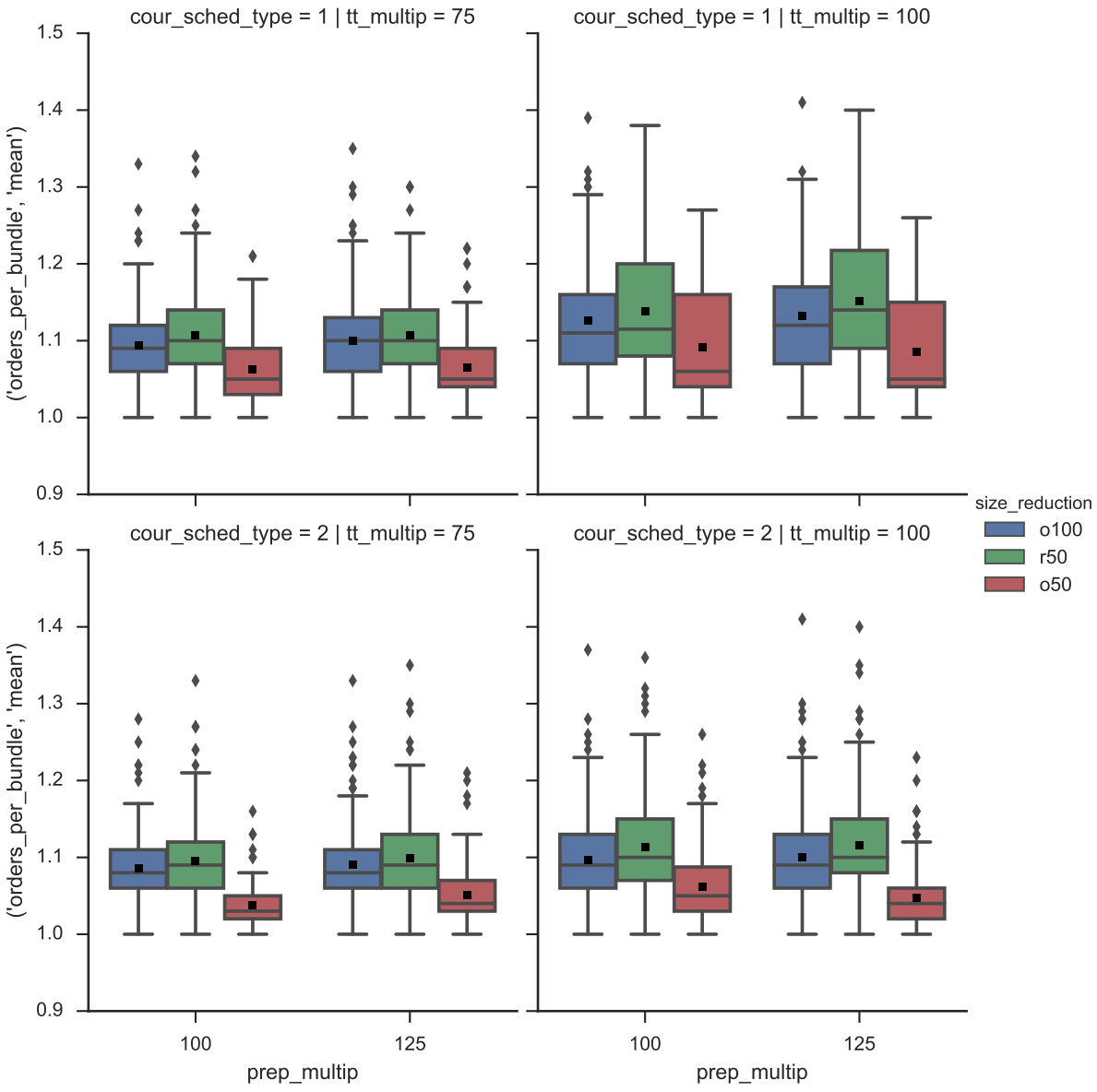


Figure B.7: Mean orders per bundle, across different instance classes

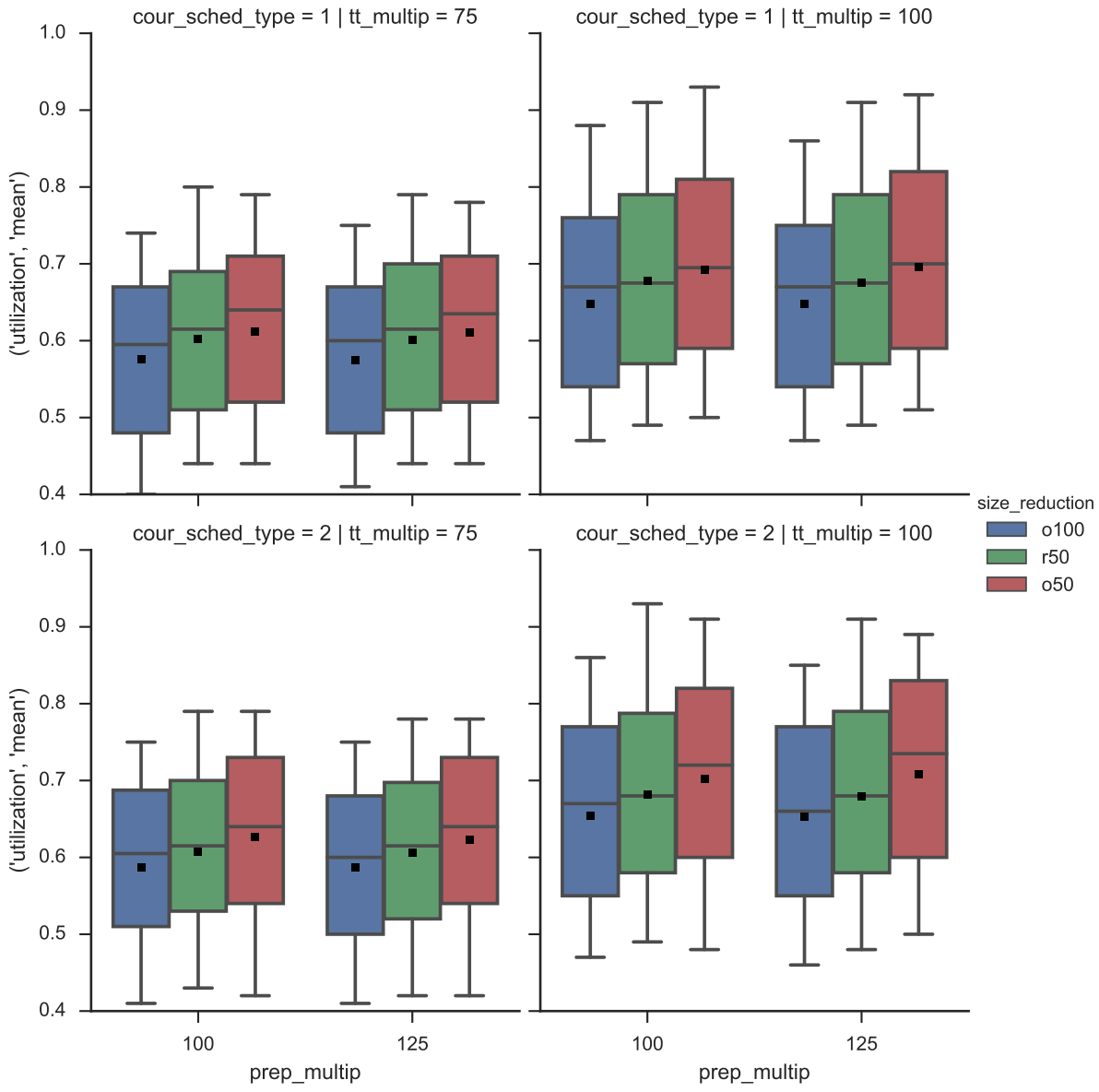


Figure B.8: Courier utilization mean, across different instance classes

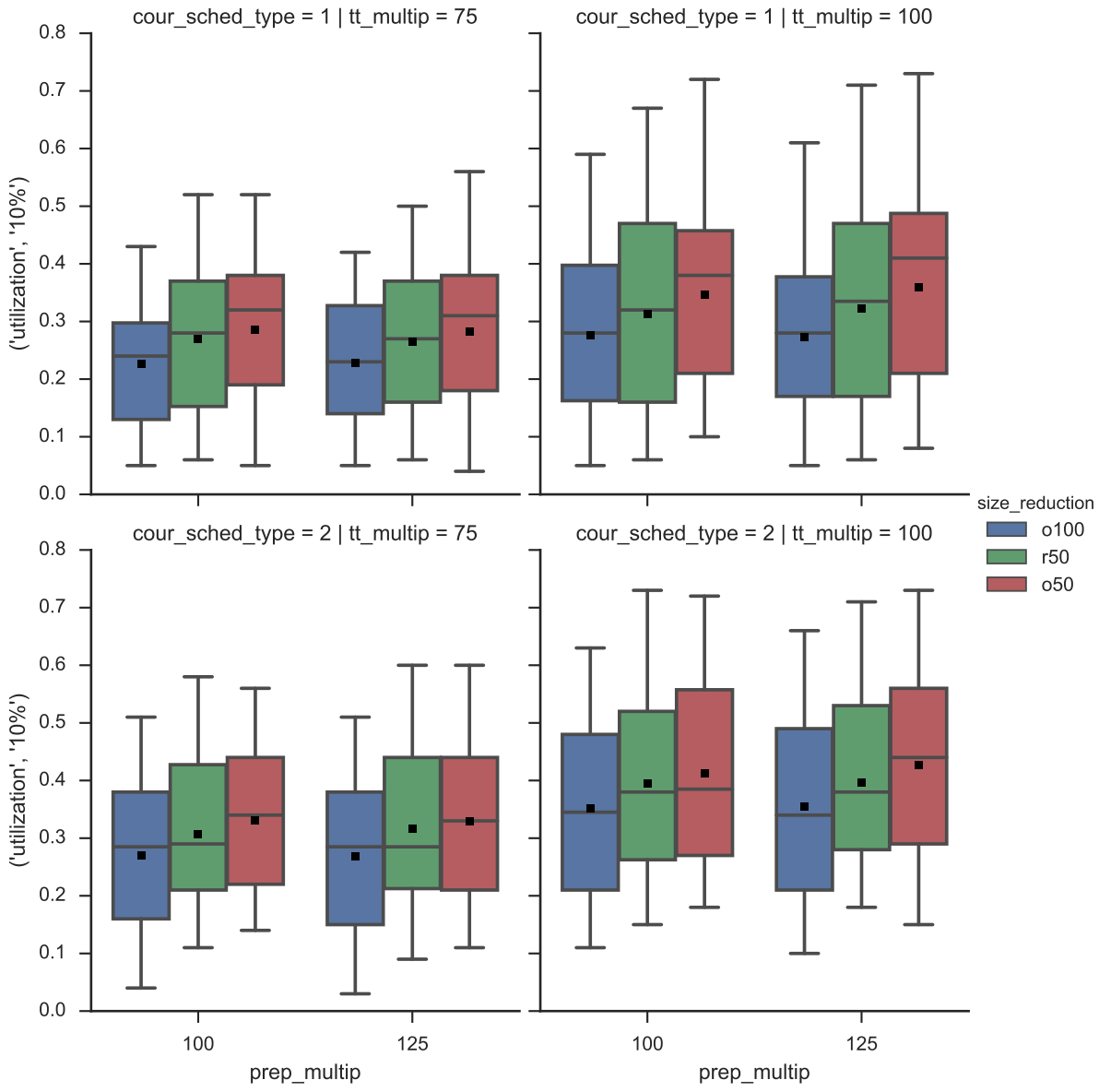


Figure B.9: Courier utilization 10th percentile, across different instance classes

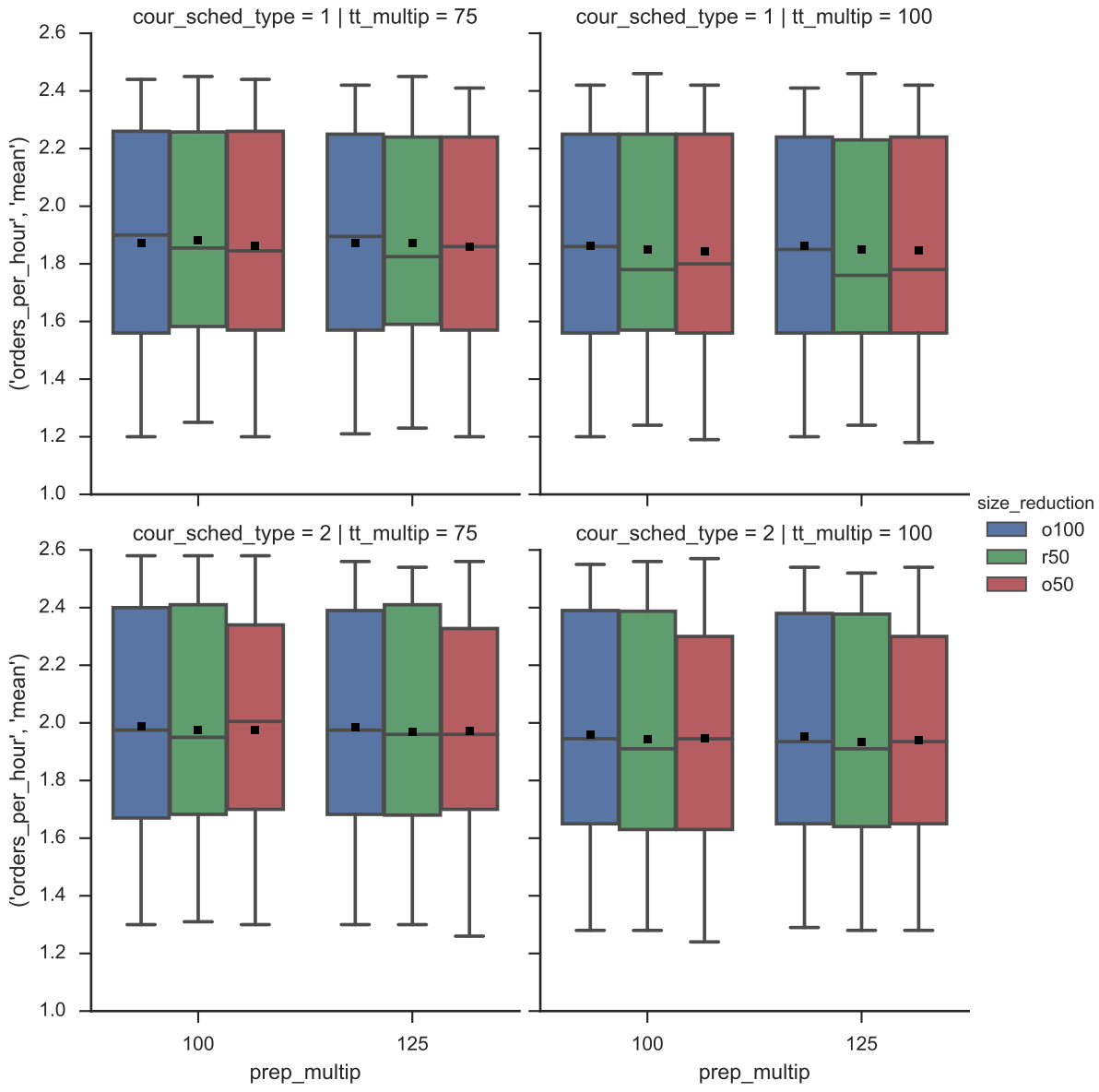


Figure B.10: Mean orders delivered per courier hour, across different instance classes

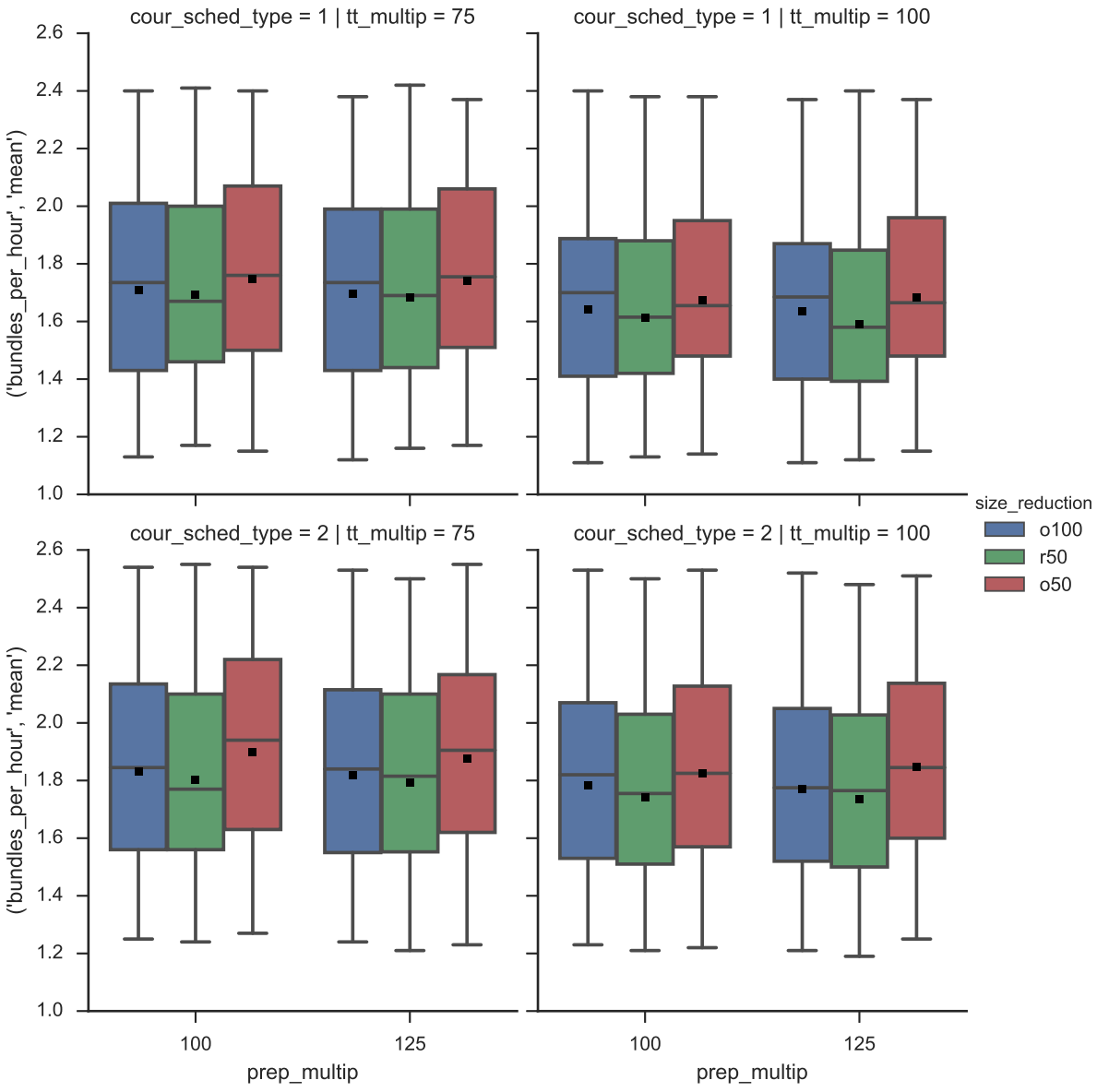


Figure B.11: Mean bundles per courier hour, across different instance classes

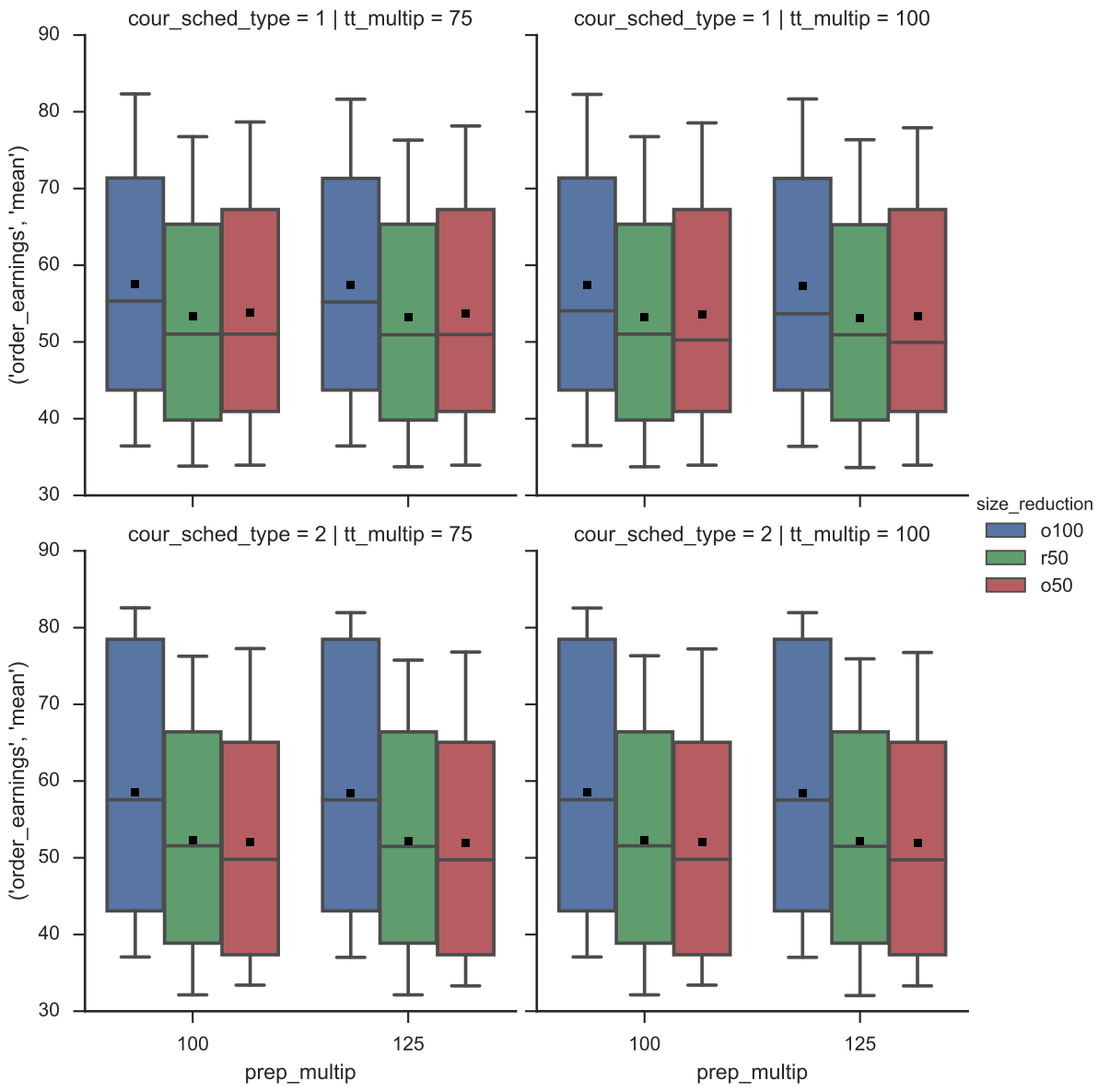


Figure B.12: Average order earnings per courier, across different instance classes

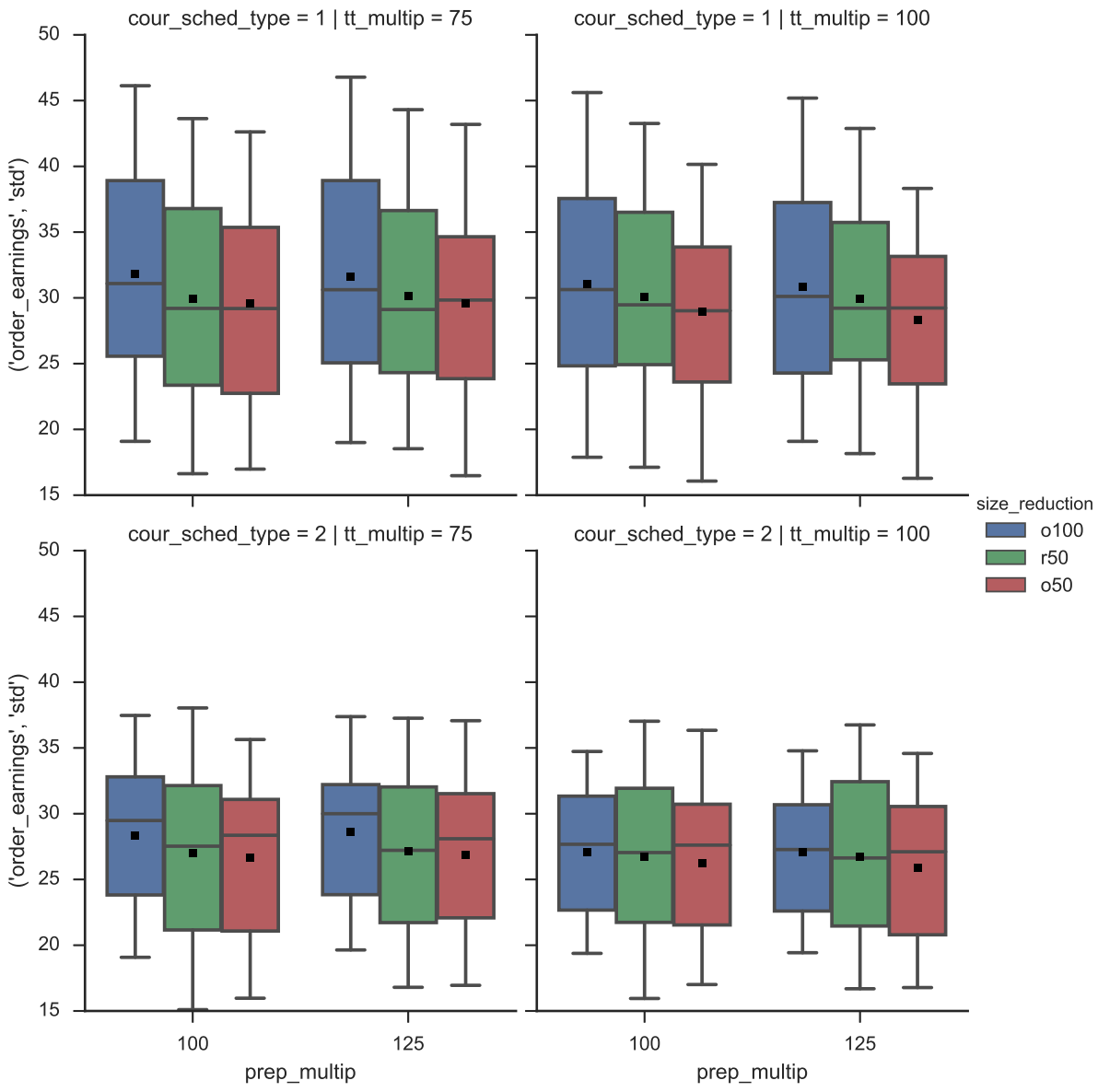


Figure B.13: Standard deviation of order earnings per courier, across different instance classes

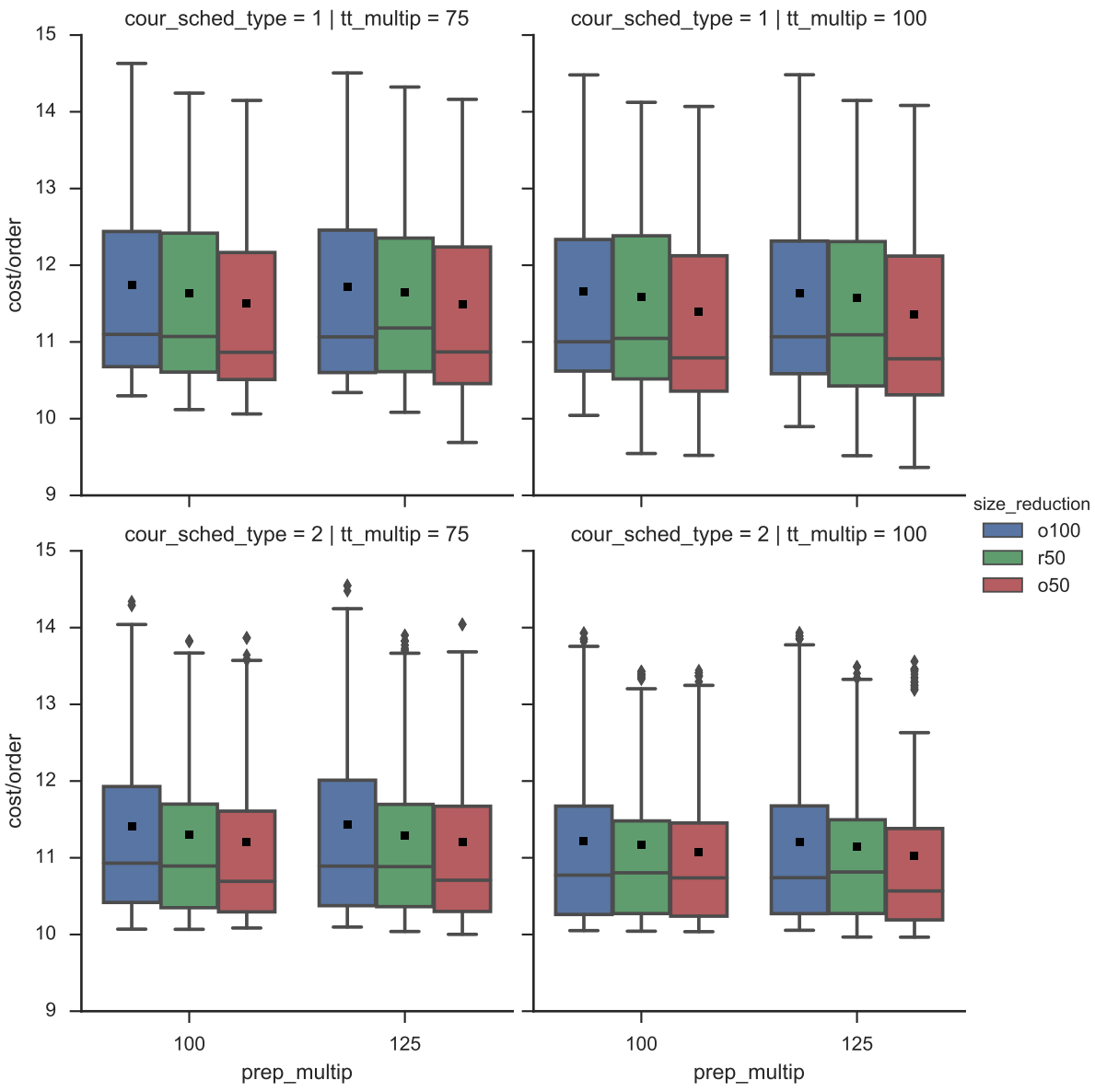


Figure B.14: Average total cost per order placed, across different instance classes

B.6 Structural properties and algorithm performance

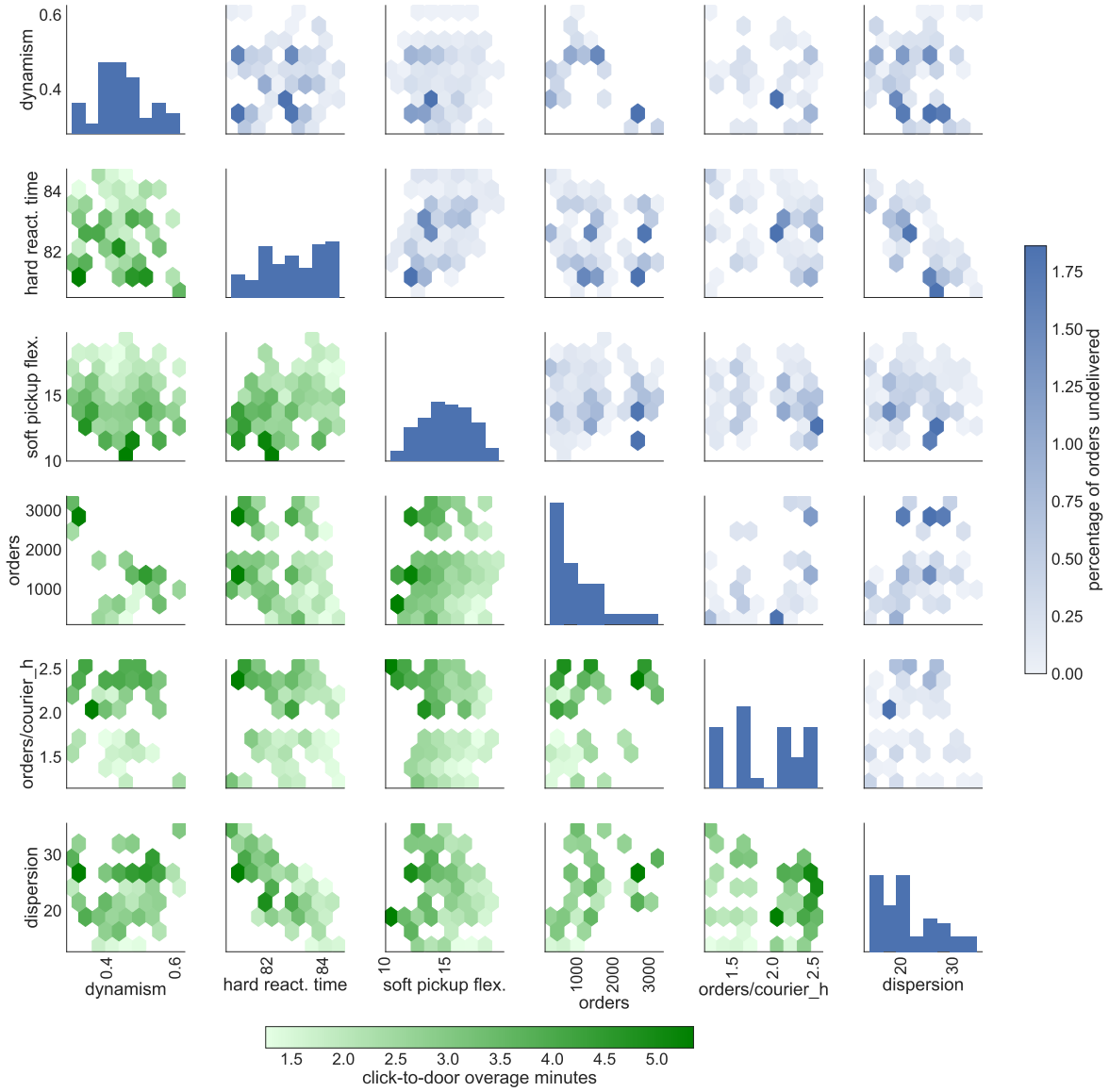


Figure B.15: Percentage of orders undelivered and click-to-door overage, and their interaction with key instance features

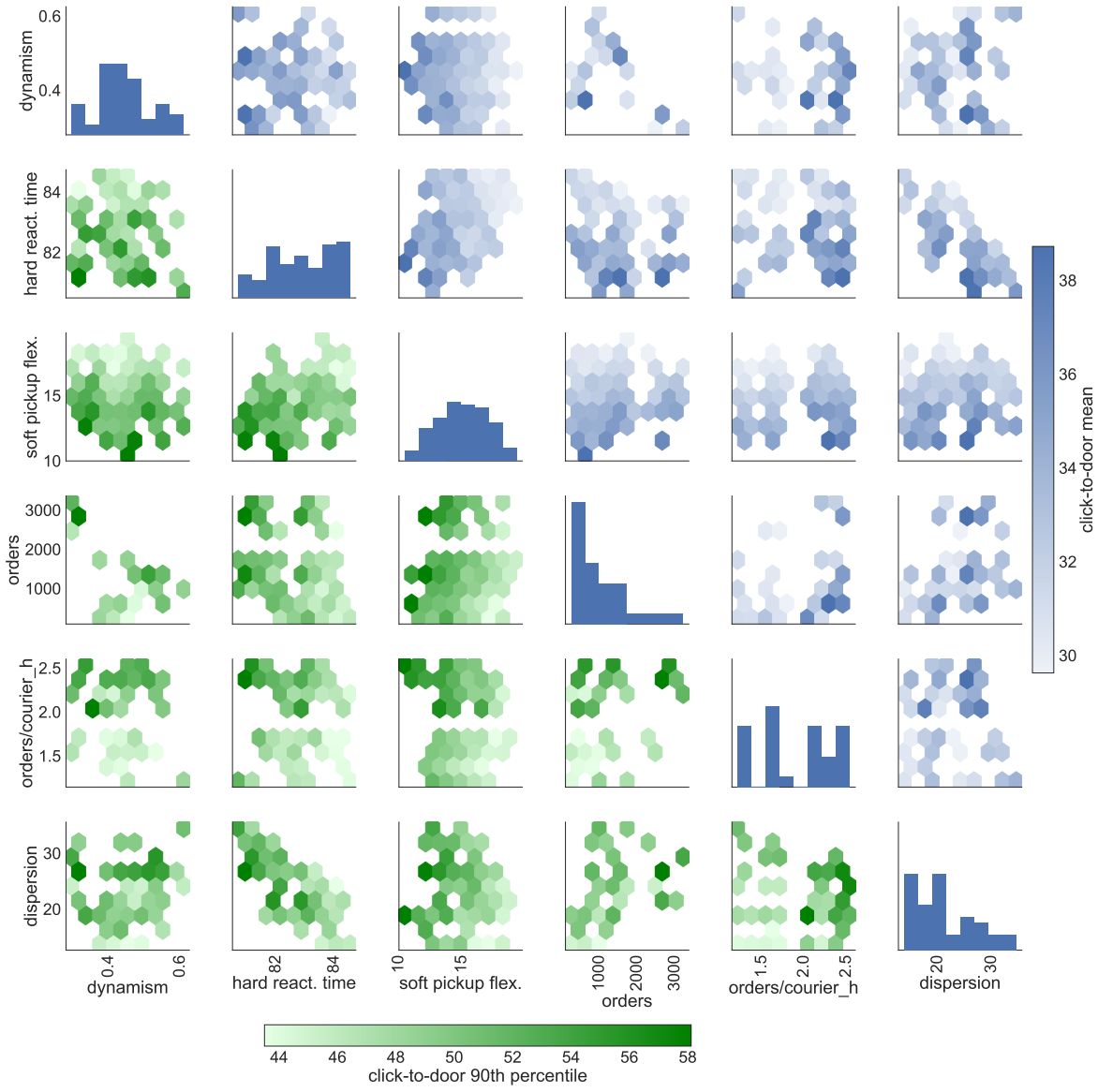


Figure B.16: Click-to-door mean and 90th percentile, and their interaction with key instance features

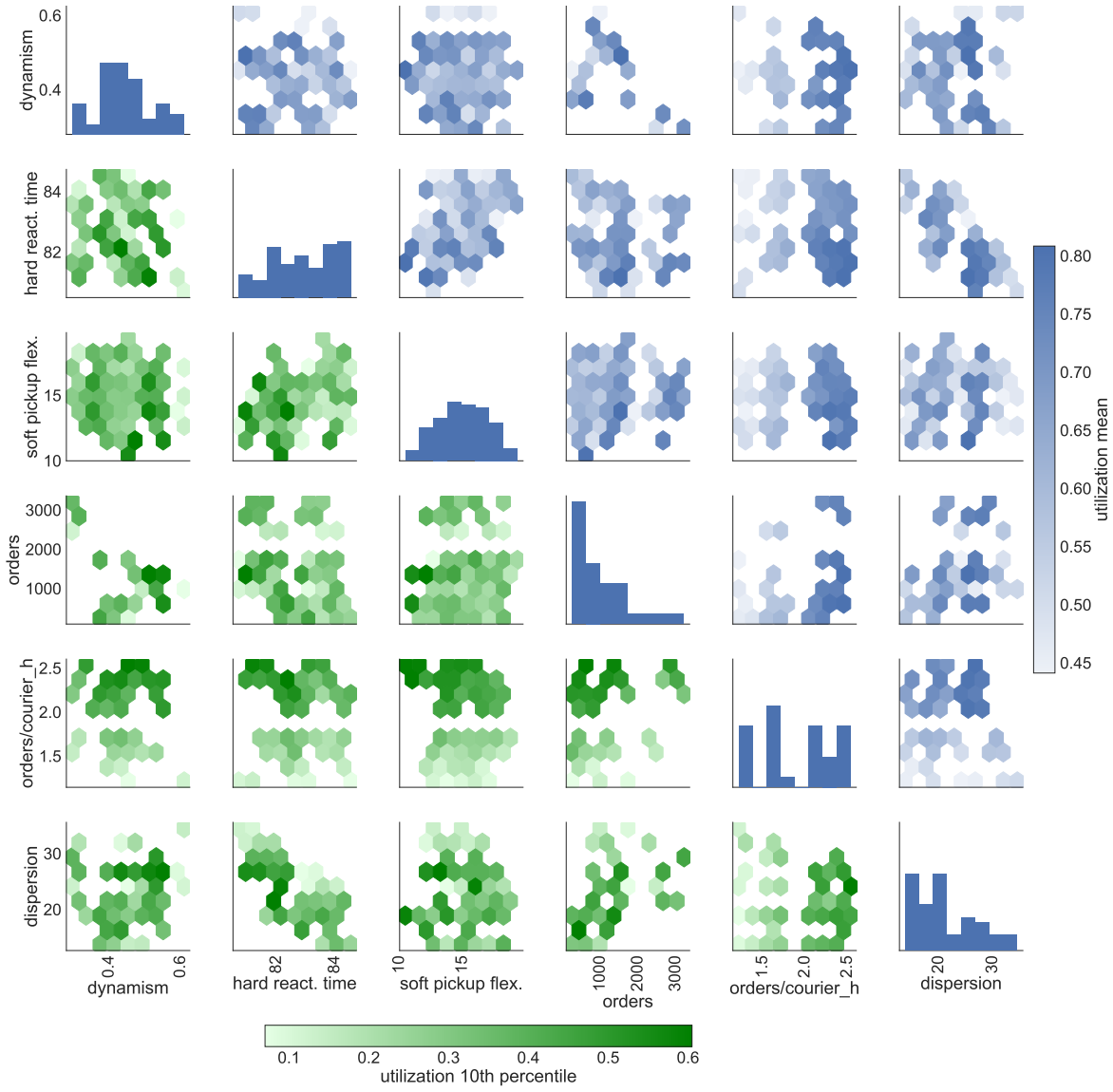


Figure B.17: courier utilization mean and 10th percentile, and their interaction with key instance features

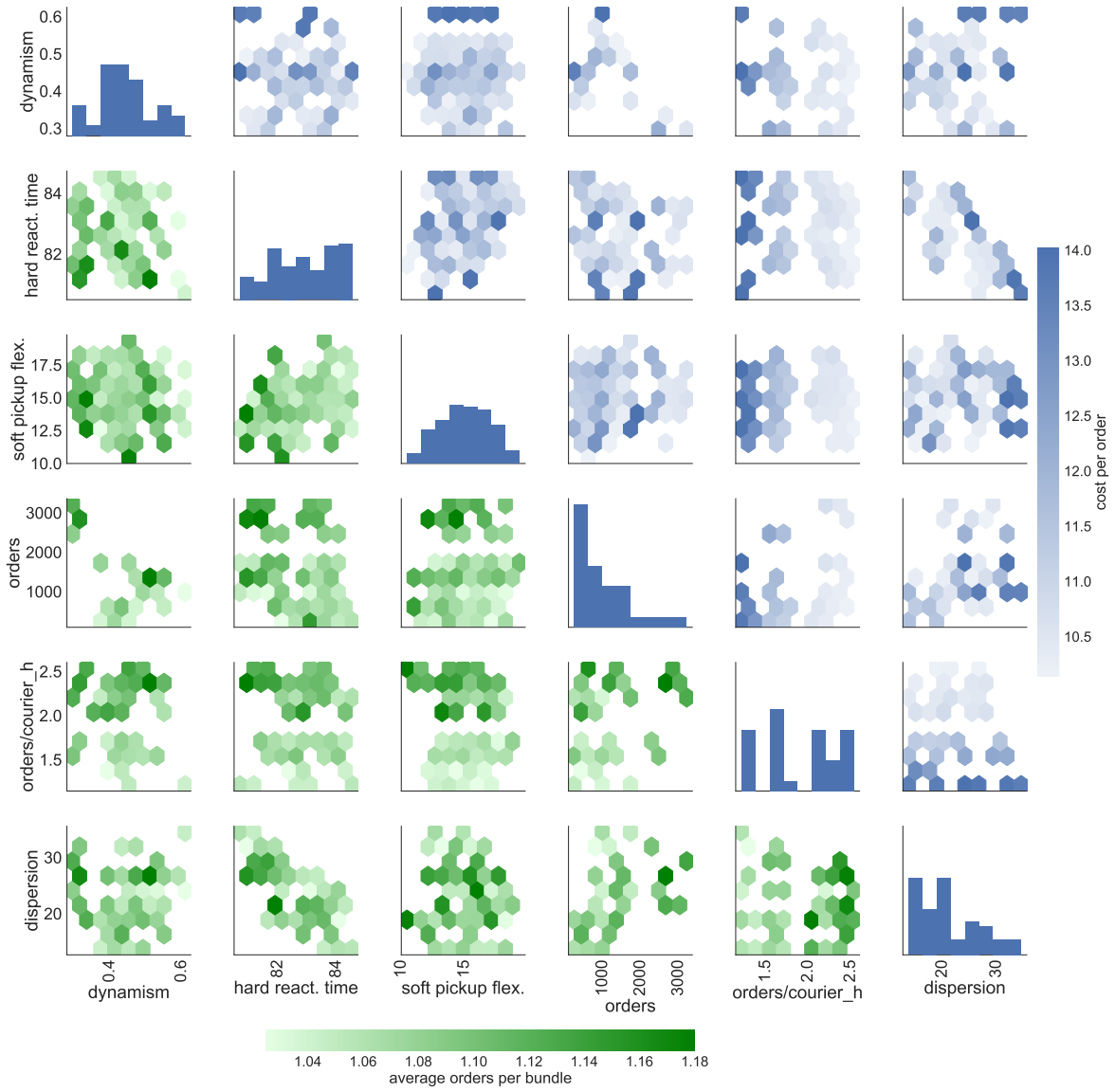


Figure B.18: cost per courier hour and average number of orders per bundle, and their interaction with key instance features

B.7 Value of key algorithmic features

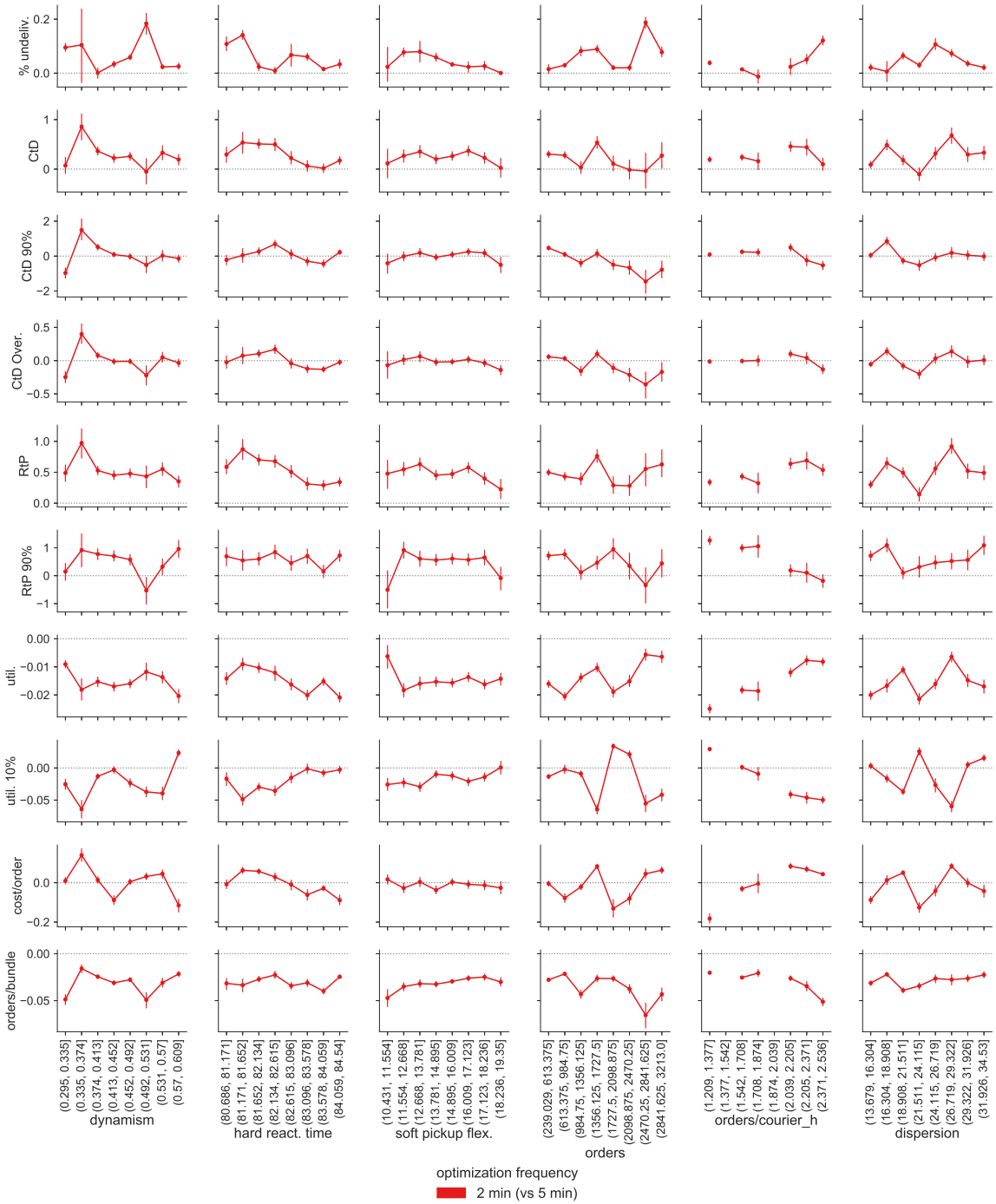


Figure B.19: Performance difference of algorithm with more frequent optimizations (2 minutes, as opposed to default 5 minutes) vs instance characteristics

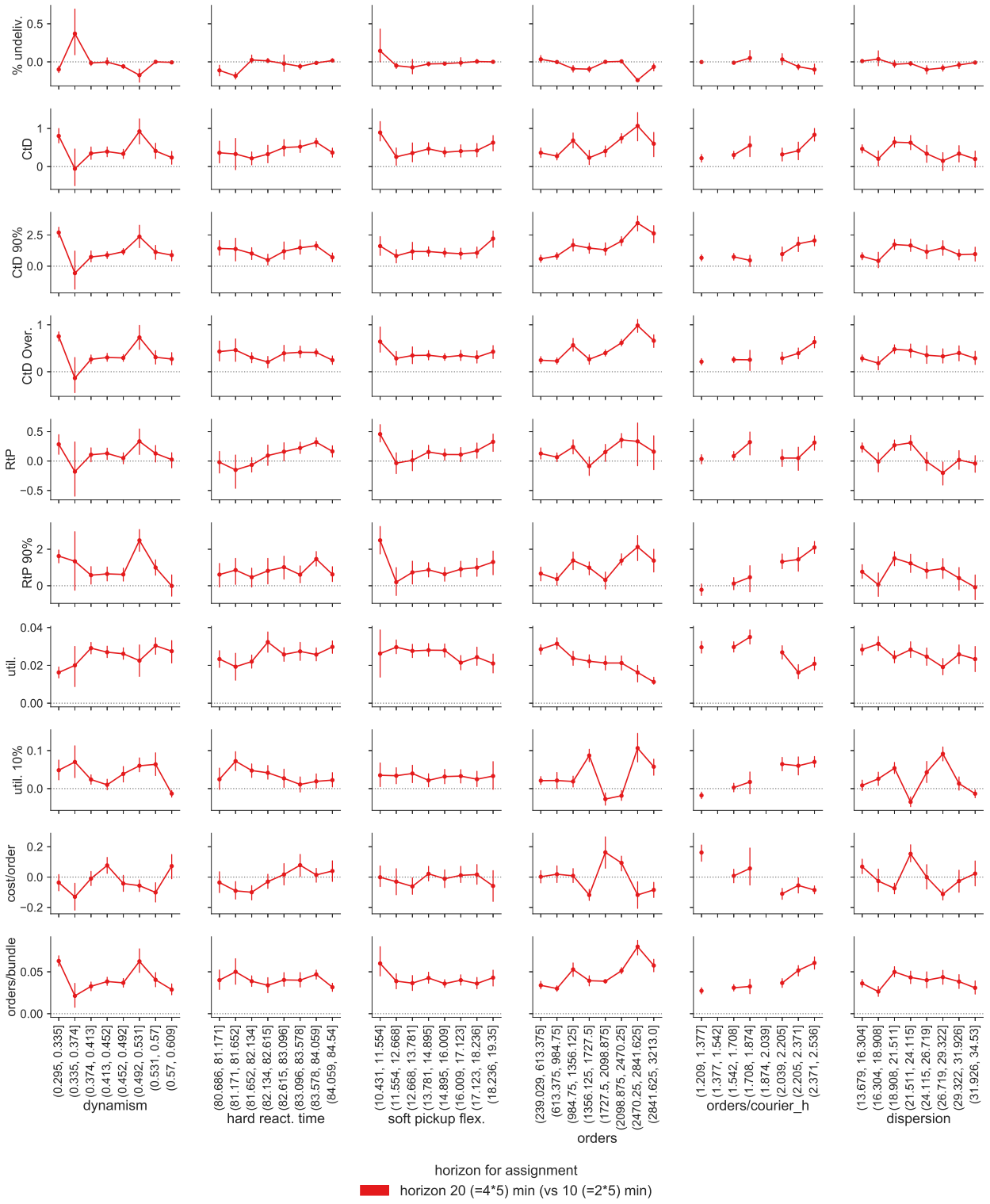


Figure B.20: Performance difference of algorithm with a 20 minute horizon for assignment of orders (as opposed to default 10 minute horizon) vs instance characteristics

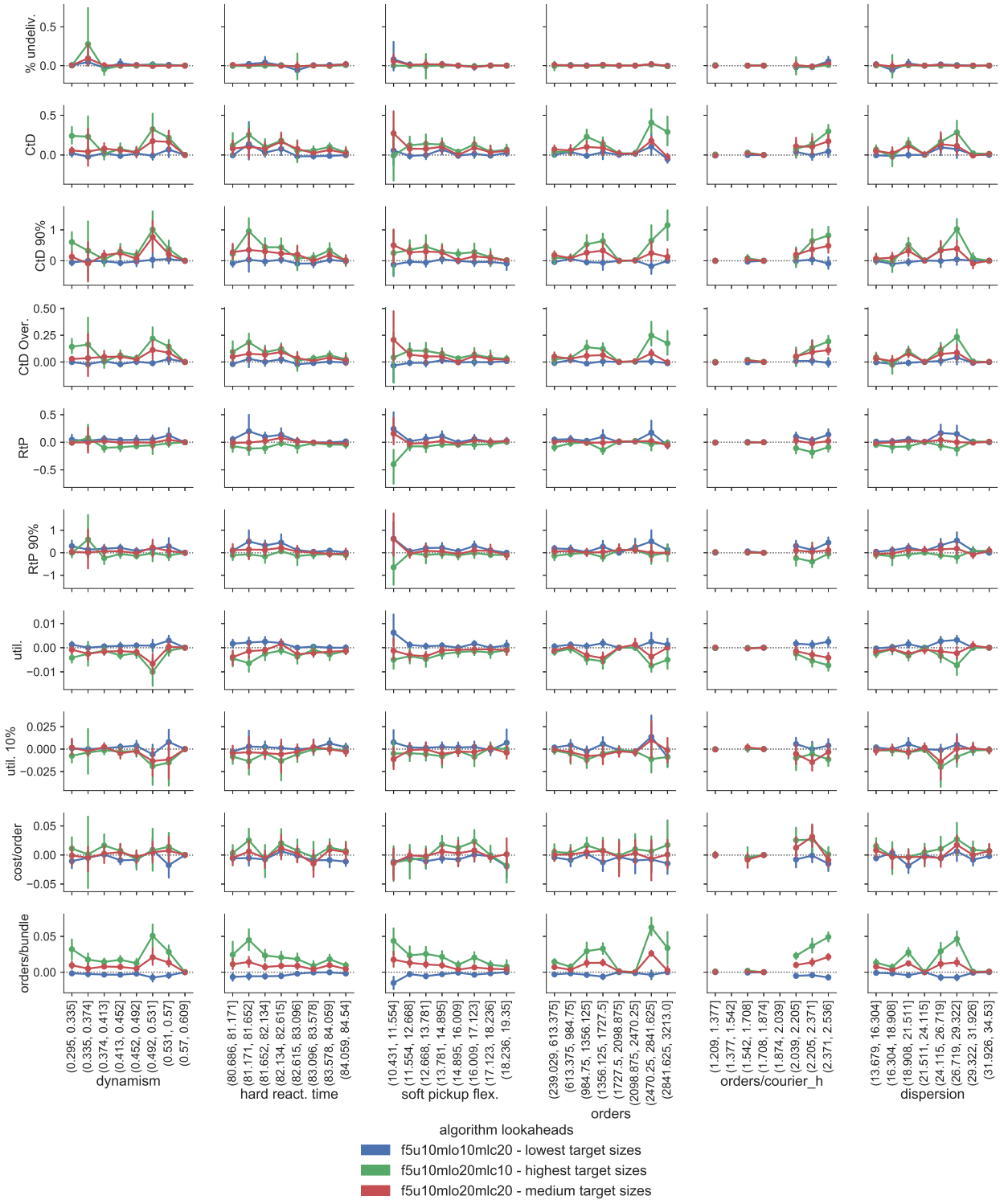


Figure B.21: Performance difference of algorithm setting alternative lookaheads for bundle targets (default is 10 minutes for both orders and couriers) vs instance characteristics

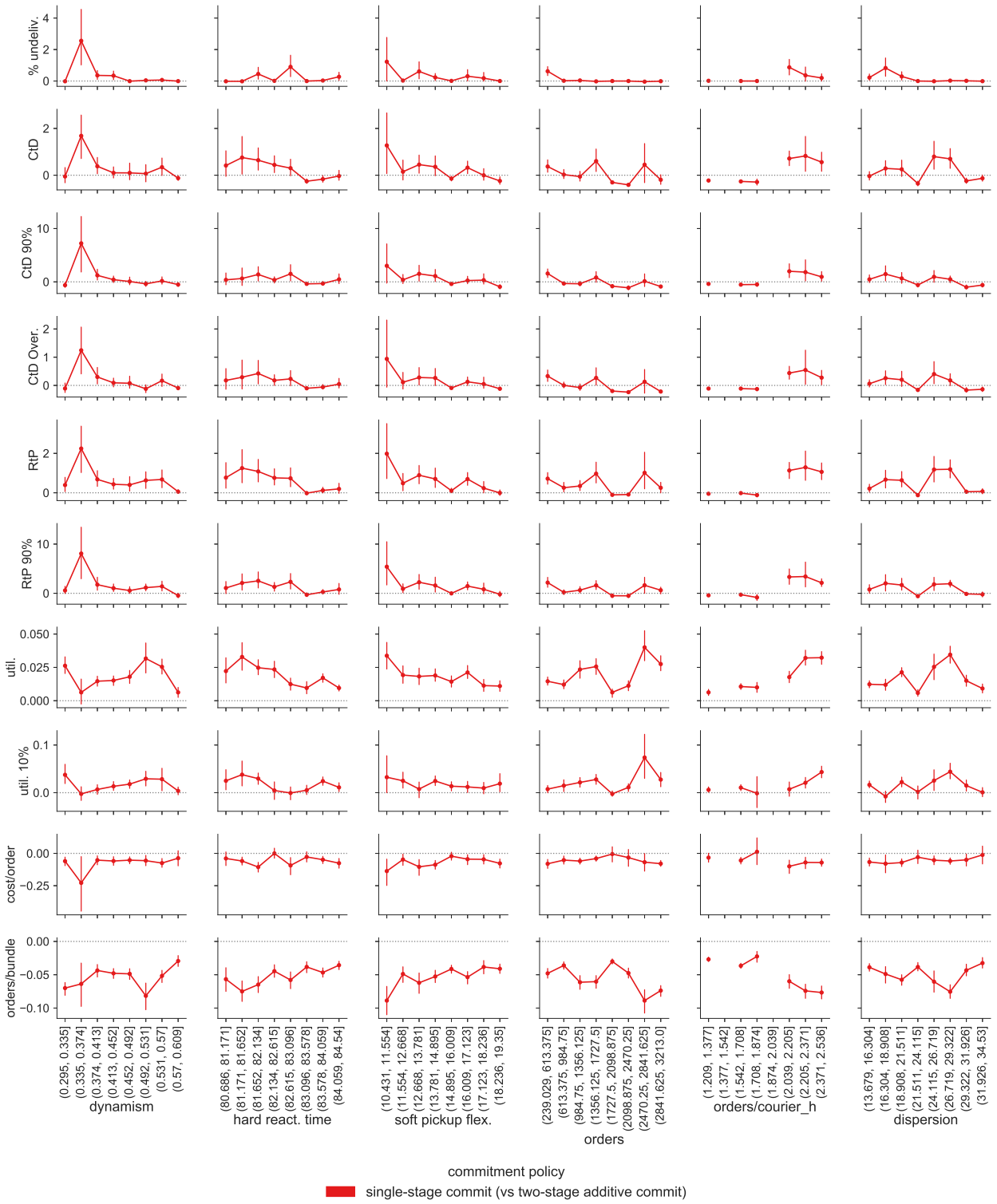


Figure B.22: Performance difference of algorithm with single-stage commitment rule (as opposed to default two-stage additive rule) vs instance characteristics

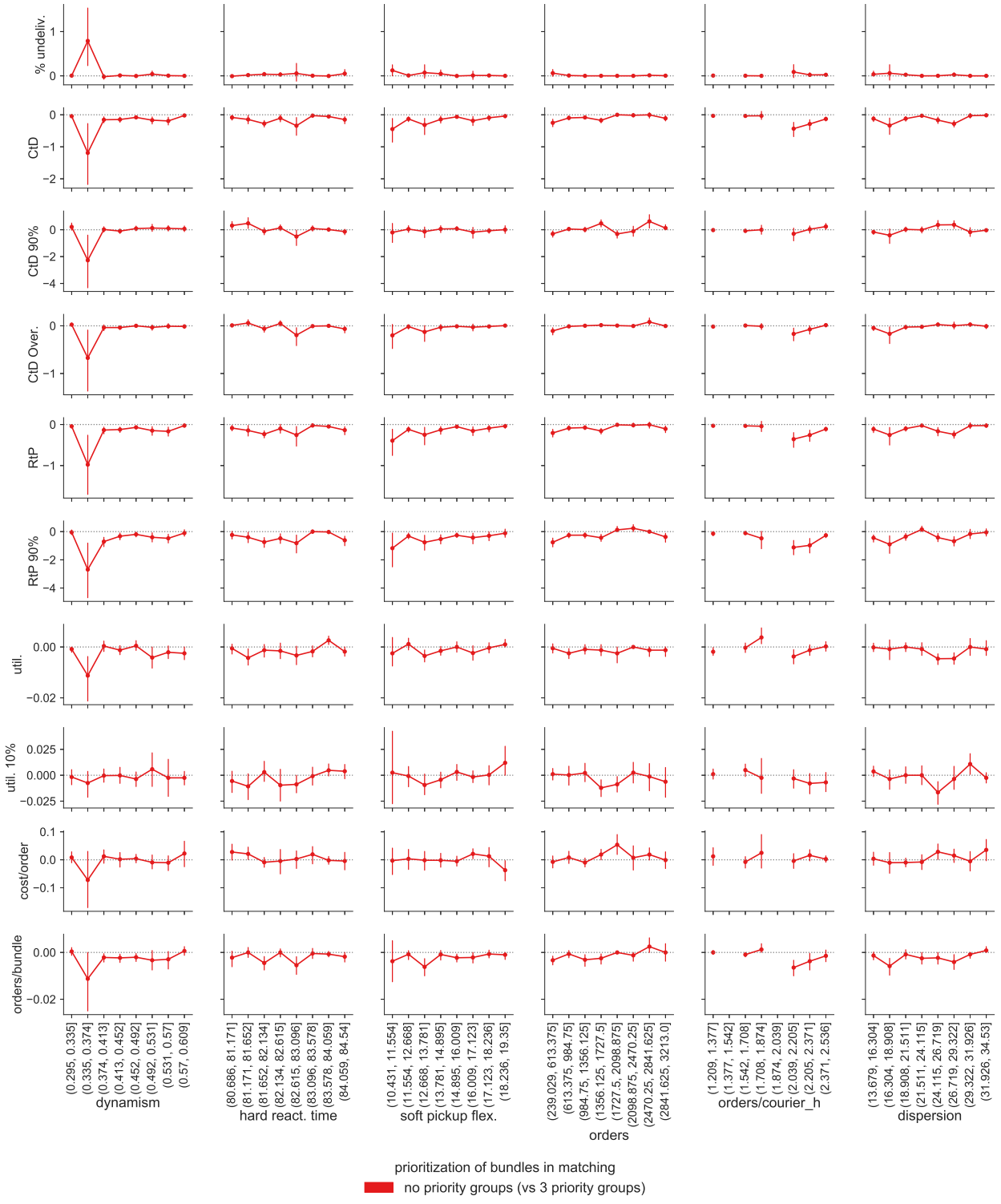


Figure B.23: Performance difference of algorithm solving one matching per optimization run (instead of solving three matchings based on order priority) vs instance characteristics

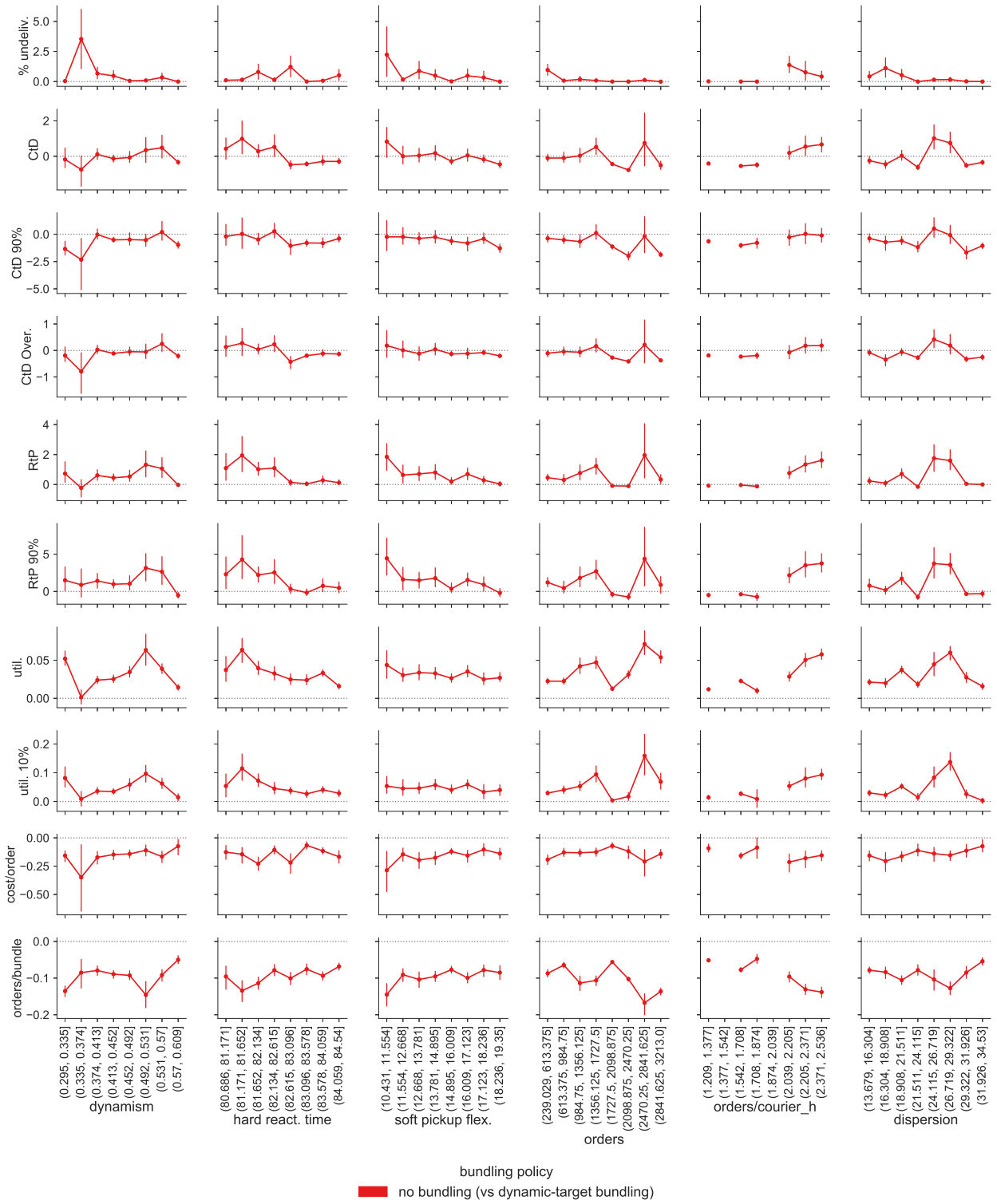


Figure B.24: Performance difference of algorithm that completely forbids bundling (as opposed to default with dynamic bundling intensity) vs instance characteristics

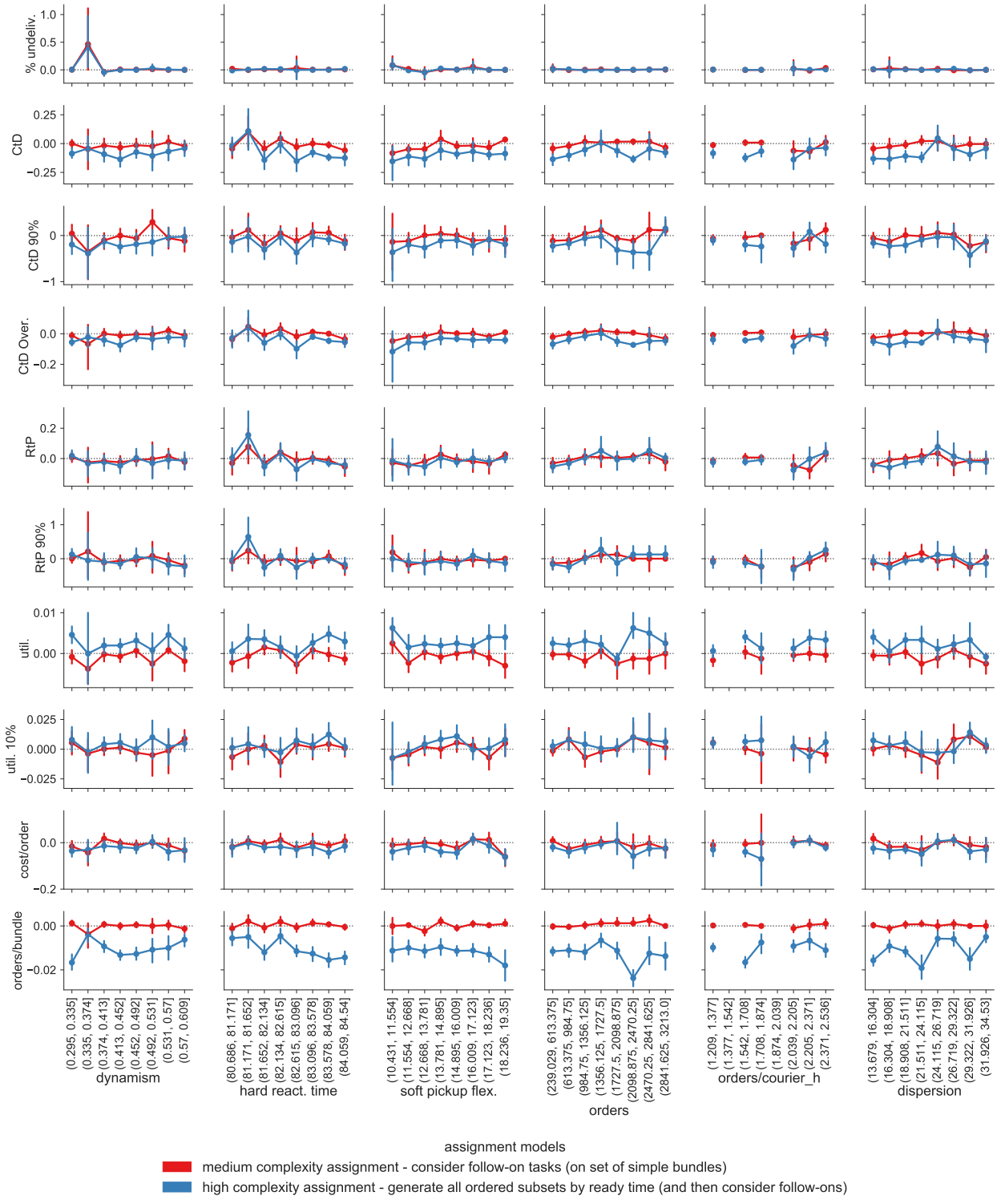


Figure B.25: Performance difference of algorithm with increasingly complex assignment models (as opposed a matching linear program) vs instance characteristics

REFERENCES

[1] "Quarterly retail e-commerce sales 2nd quarter 2017," US Census Bureau, Press Release CB17-132, 2017, Available at: https://www.census.gov/retail/mrts/www/data/pdf/ec_current.pdf. Accessed: 11-11-2017.

[2] M. Farber, *Consumers are now doing most of their shopping online*, http://fortune.com/2016/06/08/online-shopping-increases/, Accessed: 2016-07-24, 2016.

[3] UPS and comScore, *UPS Pulse of the Online Shopper*, https://solvers.ups.com/assets/2016_UPS_Pulse_of_the_Online_Shopper.pdf, Jun. 2016.

[4] M. Joerss, J. Schröder, F. Neuhaus, C. Klink, and F. Mann, "Parcel delivery: The future of last mile," McKinsey & Company, Travel, Transport and Logistics, Market report, 2016, Available at: https://www.mckinsey.com/industries/travel-transport-and-logistics/our-insights/how-customer-demands-are-reshaping-last-mile-delivery. Accessed: 11-11-2017.

[5] M. Savelsbergh and T. V. Woensel, "50th anniversary invited article — City Logistics: Challenges and Opportunities," *Transportation Science*, vol. 50, no. 2, pp. 579–590, 2016. eprint: http://dx.doi.org/10.1287/trsc.2016.0675.

[6] W. B. Cassidy, *Facing low demand and new rules, truckers pile into last-mile logistics*.

[7] T. Netzer, J. Krause, L. Hausmann, F. Bauer, and T. Ecker, "The urban delivery bet: Usd 5 billion in venture capital at risk?" McKinsey & Company, Travel, Transport and Logistics, Market report, 2017, Available at: https://www.mckinsey.com/industries/travel-transport-and-logistics/our-insights/how-will-same-day-and-on-demand-delivery-evolve-in-urban-markets. Accessed: 11-11-2017.

[8] K. Korosec, *Volvo's Solution for the Package Theft Epidemic: Your Car's Trunk*, http://fortune.com/2016/05/10/volvo-urb-it-delivery/, Accessed: 2017-02-27, May 2016.

[9] D. Etherington, *Daimler begins testing Smart car trunk delivery service with DHL*, https://techcrunch.com/2016/09/02/daimler-begins-testing-smart-car-trunk-delivery-service-with-dhl, Accessed: 2017-02-27, Sep. 2016.

[10] Audi, *Audi, DHL and Amazon deliver convenience*, https://www.audiusa.com/newsroom/news/press-releases/2015/04/audi-dhl-and-amazon-deliver-convenience, Accessed: 2016-07-24, 2015.

[11] G. Dantzig and J. Ramser, "The Truck Dispatching Problem," *Management Science*, vol. 6, pp. 80–91, 1959.

[12] G. Laporte, "Fifty Years of Vehicle Routing," *Transportation Science*, vol. 43, no. 4, pp. 408–416, 2009.

[13] M. Savelsbergh, "Local search for routing problems with time windows," *Annals of Operations Research*, vol. 4, pp. 285–305, 1986.

[14] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations Research*, vol. 35, no. 2, pp. 254–265, 1987.

[15] R. Baldacci, A. Mingozzi, and R. Roberti, "Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints," *European Journal of Operational Research*, vol. 218, no. 1, pp. 1–6, 2012.

[16] T. Vidal, T. Crainic, M. Gendreau, and C. Prins, "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows," *Computers and Operations Research*, vol. 40, pp. 475–489, 2013.

[17] T. Bektaş, G. Erdoğan, and S. Røpke, "Formulations and Branch-and-Cut Algorithms for the Generalized Vehicle Routing Problem," *Transportation Science*, vol. 45, pp. 299–316, 2011.

[18] L. Moccia, J.-F. Cordeau, and G. Laporte, "An incremental tabu search heuristic for the generalized vehicle routing problem with time windows," *Journal of the Operational Research Society*, vol. 63, no. 2, pp. 232–244, 2012.

[19] N. Azi, M. Gendreau, and J.-Y. Potvin, "A dynamic vehicle routing problem with multiple delivery routes," *Annals of Operations Research*, vol. 199, no. 1, pp. 103–112, 2012.

[20] E. Bakker, *The on-demand meal delivery report: Sizing the market, outlining the business models, and determining the future market leaders*, http://read.bi/2bU7EuD, Accessed: 2016-12-01, 2016.

[21] G. Berbeglia, J. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," *European Journal of Operational Research*, vol. 202, no. 1, pp. 8–15, 2010.

[22] H. Psaraftis, M. Wen, and C. Kontovas, "Dynamic vehicle routing problems: Three decades and counting," *Networks*, vol. 67, no. 1, pp. 3–31, 2016.

[23] G. Ghiani, E. Manni, and B. Thomas, "A comparison of anticipatory algorithms for the dynamic and stochastic traveling salesman problem," *Transportation Science*, vol. 46, no. 3, pp. 374–387, 2012.

[24] F. Ferrucci and S. Bock, "Real-time control of express pickup and delivery processes in a dynamic environment," *Transportation Research Part B: Methodological*, vol. 63, pp. 1 –14, 2014.

[25] R. R. van Lon, E. Ferrante, A. E. Turgut, T. Wenseleers, G. V. Berghe, and T. Holvoet, "Measures of dynamism and urgency in logistics," *European Journal of Operational Research*, vol. 253, no. 3, pp. 614–624, 2016.

[26] Reuters, *How Amazon is making package delivery even cheaper*, `http://fortune.com/2016/02/18/amazon-flex-deliveries/`, 2016.

[27] M. K. Lee, D. Kusbit, E. Metsky, and L. Dabbish, "Working with machines: The impact of algorithmic and data-driven management on human workers," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15, Seoul, Republic of Korea: ACM, 2015, pp. 1603–1612, ISBN: 978-1-4503-3145-6.

[28] D. Reyes, M. Savelsbergh, and A. Toriello, "Vehicle routing with roaming delivery locations," *Transportation Research Part C: Emerging Technologies*, vol. 80, pp. 71–91, 2017.

[29] V. Pillac, M. Gendreau, C. Guéret, and A. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[30] C. Archetti, D. Feillet, and M. Speranza, "Complexity of routing problems with release dates," *European Journal of Operational Research*, vol. 247, no. 3, pp. 797–803, 2015.

[31] D. Reyes, A. L. Erera, and M. W. Savelsbergh, "Complexity of routing problems with release dates and deadlines," *European Journal of Operational Research*, 2017.

[32] Wal-Mart Stores, Inc., *Walmart 2016 global responsibility report*, `http://corporate.walmart.com/2016grr`, 2016.

[33] United Parcel Service of America, Inc., *UPS 2015 corporate sustainability report*, https://sustainability.ups.com/media/ups-pdf-interactive/UPS_2015_CSR.pdf, Jul. 2016.

[34] J. Biggs, *Cardrops Is A Service That Puts Stuff You Order Into The Trunk Of Your Car. Yeah. Really.* https://techcrunch.com/2012/10/29/cardrops-is-a-service-that-puts-stuff-you-order-into-the-trunk-of-your-car-yeah-really/, Accessed: 2017-03-30, 2012.

[35] A. Davies, *Volvo Ditches the Car Key to Make Way for the Future*, https://www.wired.com/2016/02/volvo-kills-the-car-key-to-make-way-for-the-future/, Accessed: 2017-03-30, 2016.

[36] Volvo Cars, *Volvo cars demonstrates the potential of connected cars with deliveries direct to people's cars*, Press Release, https://www.media.volvocars.com/global/en-gb/media/pressreleases/139114/volvo-cars-demonstrates-the-potential-of-connected-cars-with-deliveries-direct-to-peoples-cars, 2014.

[37] F. Gleyo, "Volvo in-car delivery will send your shopping goods straight to your car trunk," *Tech Times*, 2015.

[38] C. Malandraki and M. Daskin, "Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms," *Transportation Science*, vol. 26, pp. 185–200, 1992.

[39] S. Ichoua, M. Gendreau, and J.-Y. Potvin, "Vehicle dispatching with time-dependent travel times," *European Journal of Operational Research*, vol. 144, no. 2, pp. 379 –396, 2003.

[40] M. Figliozzi, "The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 3, pp. 616 –636, 2012.

[41] M. Desrochers, J. Lenstra, M. Savelsbergh, and F. Soumis, "Vehicle Routing with Time Windows: Optimization and Approximation," in *Vehicle Routing: Methods and Studies*, B. Golden and A. Assad, Eds., North-Holland: Elsevier Science Publishers B.V., 1988, pp. 65–84.

[42] A. Campbell and M. Savelsbergh, "Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems," *Transportation Science*, vol. 38, pp. 369–378, 2004.

[43] T. Feo and M. Resende, "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[44] C. Cerrone, R. Cerulli, and B. Golden, "Carousel Greedy: A Generalized Greedy Algorithm for Optimizing the Cardinality of a Set," Presented at Graphs and Optimization IX Meeting, Sirmione, Italy. http://scholar.rhsmith.umd.edu/sites/default/files/bgolden/files/carousel_greedy.pdf, 2014.

[45] D. Pisinger and S. Røpke, "A General Heuristic for Vehicle Routing Problems," *Computers and Operations Research*, vol. 34, pp. 2403–2435, 2007.

[46] M. Klapp, A. Erera, and A.Toriello, "The one-dimensional dynamic dispatch waves problem," 2015, Optimization Online 2015-03-4826.

[47] C. Hirschberg, A. Rajko, T. Schumacher, and M. Wrulich, *The changing market for food delivery*, https://www.mckinsey.com/industries/high-tech/our-insights/the-changing-market-for-food-delivery, Accessed: 2017-05-01, Nov. 2016.

[48] Morgan Stanley Research, *Is online food delivery about to get 'amazoned'?* https://www.morganstanley.com/ideas/online-food-delivery-market-expands/, Accessed: 2017-10-04, 2017.

[49] C. Dewey, *The insane $43 billion system that gets food delivered to your door*, https://www.washingtonpost.com/news/wonk/wp/2017/08/08/the-insane-43-billion-system-that-gets-food-delivered-to-your-door/, Accessed: 2017-10-03, Aug. 2017.

[50] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *European Journal of Operational Research*, vol. 223, no. 2, pp. 295 –303, 2012.

[51] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, no. Supplement C, pp. 28 –46, 2013.

[52] M. Stiglic, N. Agatz, M. Savelsbergh, and M. Gradisar, "Making dynamic ride-sharing work: The impact of driver and rider flexibility," *Transportation Research Part E: Logistics and Transportation Review*, vol. 91, no. Supplement C, pp. 190 –207, 2016.

[53] X. Wang, N. Agatz, and A. Erera, "Stable matching for dynamic ride-sharing systems," *Transportation Science*, vol. 0, no. 0, null, 0.

[54] M. A. Klapp, A. L. Erera, and A. Toriello, "The one-dimensional dynamic dispatch waves problem," *Transportation Science*, 2016.

[55] S. A. Voccia, A. M. Campbell, and B. W. Thomas, "The same-day delivery problem for online purchases," *Transportation Science*, vol. 0, no. 0, null, 0.

[56] M. A. Klapp, "Dynamic optimization for same-day delivery operations," PhD thesis, Georgia Institute of Technology, 2016.

[57] C. Archetti, M. Savelsbergh, and M. G. Speranza, "The vehicle routing problem with occasional drivers," *European Journal of Operational Research*, vol. 254, no. 2, pp. 472 –480, 2016.

[58] I. Dayarian and M. Savelsbergh, "Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders," *Optimization Online*, 2017.

[59] M. Ulmer, B. Thomas, and D. Mattfeld, "Preemptive depot returns for a dynamic same-day delivery problem," Working paper. TU Braunschweig, Germany, 2016.

[60] J. Yang, P. Jaillet, and H. Mahmassani, "Real-time multivehicle truckload pickup and delivery problems," *Transportation Science*, vol. 38, no. 2, pp. 135–148, 2004. eprint: `http://pubsonline.informs.org/doi/pdf/10.1287/trsc.1030.0068`.

[61] S. Mitrovic-Minic, R. Krishnamurti, and G. Laporte, "Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows," *Transportation Research Part B: Methodological*, vol. 38, no. 8, pp. 669 –685, 2004.

[62] S. Mitrovic-Minic and G. Laporte, "Waiting strategies for the dynamic pickup and delivery problem with time windows," *Transportation Research Part B*, vol. 38, no. 7, pp. 635–655, 2004.

[63] K. Lund, O. B. Madsen, and J. M. Rygaard, "Vehicle routing problems with varying degrees of dynamism," Department of Mathematical Modeling, The Technical University of Denmark, Tech. Rep. IMM-REP-1996-1, May 1996.

[64] A. Larsen, O. Madsen, and M. Solomon, "Partially dynamic vehicle routing—models and algorithms," *Journal of the Operational Research Society*, vol. 53, no. 6, pp. 637–646, 2002.

[65] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & operations research*, vol. 34, no. 8, pp. 2403–2435, 2007.

[66] B. Yildiz and M. Savelsbergh, "Provably high-quality solutions for the meal delivery routing problem," Georgia Institute of Technology, 2017.

[67] E. Altman, T. Jiménez, and G. Koole, "On optimal call admission control in resource-sharing system," *IEEE Transactions on Communications*, vol. 49, no. 9, pp. 1659–1668, 2001.

[68] X. Fan-Orzechowski and E. A. Feinberg, "Optimality of randomized trunk reservation for a problem with a single constraint," *Advances in applied probability*, vol. 38, no. 1, pp. 199–220, 2006.

[69] E. Carrizosa, E Conde, and M Munoz-Marquez, "Admission policies in loss queueing models with heterogeneous arrivals," *Management Science*, vol. 44, no. 3, pp. 311–320, 1998.

[70] Y. Chen, R. Levi, and C. Shi, "Revenue management of reusable resources with advanced reservations," *Production and Operations Management*, vol. 26, no. 5, pp. 836–859, 2017.

[71] K. Rajaram and C. S. Tang, "The impact of product substitution on retail merchandising," *European Journal of Operational Research*, vol. 135, no. 3, pp. 582 –601, 2001.

[72] G. Gallego and R. Phillips, "Revenue management of flexible products," *Manufacturing & Service Operations Management*, vol. 6, no. 4, pp. 321–337, 2004. eprint: https://doi.org/10.1287/msom.1040.0054.

[73] S. Koch, J. Gönsch, and C. Steinhardt, "Dynamic programming decomposition for choice-based revenue management with flexible products," *Transportation Science*, vol. 51, no. 4, pp. 1046–1062, 2017. eprint: https://doi.org/10.1287/trsc.2017.0743.

[74] J. C. Castillo, D. Knoepfle, and G. Weyl, "Surge pricing solves the wild goose chase," in *Proceedings of the 2017 ACM Conference on Economics and Computation*, ser. EC '17, Cambridge, Massachusetts, USA: ACM, 2017, pp. 241–242, ISBN: 978-1-4503-4527-9.

[75] L. Chen, A. Mislove, and C. Wilson, "Peeking beneath the hood of uber," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15, Tokyo, Japan: ACM, 2015, pp. 495–508, ISBN: 978-1-4503-3848-6.

[76] S. Banerjee, R. Johari, and C. Riquelme, "Dynamic pricing in ridesharing platforms," *SIGecom Exch.*, vol. 15, no. 1, pp. 65–70, Sep. 2016.

[77] E. Angelelli, M. Grazia Speranza, and M. W. Savelsbergh, "Competitive analysis for dynamic multiperiod uncapacitated routing problems," *Networks*, vol. 49, no. 4, pp. 308–317, 2007.

[78] P. Jaillet and X. Lu, "Online traveling salesman problems with rejection options," *Networks*, vol. 64, no. 2, pp. 84–95, 2014.

[79] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

VITA

Luis Damián Reyes Rodríguez was born in San Salvador, El Salvador, in 1990. During his middle and high school years, Damián was a student and instructor at the University of El Salvador's Talented Youth Program, as well as member of the national mathematical olympic team in various international competitions. In 2009 he received a scholarship from the Government of El Salvador and enrolled at the University of Illinois at Urbana-Champaign, where he majored in Mathematics and Economics, and took part in the Campus Honors Program. After graduating Summa Cum Laude in 2013, Damián worked for the El Salvador Port Authority as an Economic and Financial Analyst at the Business Intelligence Unit, supporting an array of infrastructure investment planning and operational improvement projects for the network of national seaports, airports and air navigation systems. In 2014, he enrolled as a doctoral student at the School of Industrial and Systems Engineering of the Georgia Institute of Technology, following the Supply Chain and Logistics concentration. Under the supervision of Dr. Martin Savelsbergh and Dr. Alan Erera, most of his research to this date has focused on understanding and developing appropriate optimization technologies for meal delivery systems. In addition to last-mile logistics, Damián's research interests also include disaster and humanitarian logistics, a topic of great importance for his country of birth.

Since 2014, Damián has made a common life with Kelly Ventura, a psychologist by training currently working on her masters' thesis in the Critical Pedagogy program at the University of Buenos Aires. After completion of Damián's dissertation, they are moving to Seattle, where he has accepted a position as research scientist at Amazon.